Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Distributed Mobile Graphics

by

Jiří Danihelka

A thesis submitted to
the Faculty of Electrical Engineering, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

PhD program: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering

Prague, 2015

Thesis Supervisor:
Jiří Žára
Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University in Prague

Acknowledgments

Acknowledgments from the most important related publications:

[A.1] This research has been partially supported by the Technology Agency of the Czech Republic under research program TE01020415 (V3C – Visual Computing Competence Center) and by Microsoft Czech Republic.

[A.2] This research has been partially supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS10/291/OHK3/3T/13, the research program LC-06008 (Center for Computer Graphics), Microsoft Innovation Cluster for Embedded Software (ICES) and by Vodafone Foundation Czech Republic.

[A.3] This project was partially sponsored by Microsoft Czech Republic and the CTU FEE Interoperability Lab. Many thanks also belong to researchers in the Microsoft Research Redmond Connections and Networking Research Group, especially Dr. Judith Bishop, Dr. Victor Bahl and Dr. Stefan Saroiu, for initial project consultations. This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS10/291/OHK3/3T/13. The research was supported by the grant No. DF12P01OVV028 of the Ministry of Culture of the Czech Republic.

[A.4] The project has been sponsored by the Microsoft Innovation Cluster for Embedded Software (ICES). We thank Abouzar Noori for his initial contribution to the project and the anonymous reviewers for their inspiring and insightful feedback.

[A.5] This research has been partially supported by the MSMT under the research program MSM 6840770014, the research program LC-06008 (Center for Computer Graphics) and by Vodafone Foundation Czech Republic.

Contents

List of Abbreviations

2D	two-dimensional
3D	three-dimensional
AML	Avatar Markup Language
API	application programming interface
ARM	Advanced RISC Machines
BTS	Base Transceiver Station
CGA	Computer Generated Architecture
CPU	central processing unit
ECA	embodied conversational agent
ECAF	Authoring Language for Embodied Conversational Agents
FAP	Facial Animation Parameters
ES	embedded systems
FP	feature point
FPS	frames per second
FPU	Floating-Point Unit
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Global System for Mobile Communications
HSTP	Hyperspeech Transfer Protocol
HTC	High Tech Computers (Taiwan manufacturer of portable devices)
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organization
Java ME	Java Micro Edition
JIT	Just-In-Time compilation
JPEG	Joint Photographic Experts Group - an image file format

LCD	liquid crystal display
LOD	Level-Of-Detail – a technique for showing simpler models for far-away objects to speed up rendering
LTE	Long Term Evolution – a mobile communication standard
MPEG	Motion Picture Experts Group
MS	Microsoft
NPR	non-photorealistic rendering
OpenGL	Open Graphics Library
OpenGL ES	Open Graphics Library for Embedded Systems
OS	operating system
PC	personal computer
P2P	peer-to-peer
PDA	personal digital assistant
PNG	Portable Network Graphics - an image file format
RAM	random access memory
RGB	red-green-blue color space
RISC	Reduced Instruction Set CPU
ROM	Read-Only Memory
qVGA	quarter video graphics array resolution – 320×240 or 240×320 pixels
SDK	software development kit
SQL	Structured Query Language
SMIL-Agent	Synchronized Multichannel Integration Language for Synthetic Agents
SVGA	super video graphics array
UI	user interface
URL	Universal Resource Locator - an internet link
VGA	video graphics array
VRML	Virtual Reality Markup Language
WiDi	Wireless Display
WiMAX	Worldwide Interoperability for Microwave Access
WWTW	World-Wide Telecom Web
WWW	World-Wide Web
XML	Extensible Markup Language

Abstract

Mobile networking technologies are the most ubiquitously spread among mankind and with the technological advances of mobile clients are becoming a prime target for innovative 3D graphics applications.

Our research, presented in this thesis, focuses on new methods of reducing polygonal models and other commonly used graphical structures in order to bring 3D computer graphics to devices with limited processor speeds and memory, such as mobile phones. These environments bring new challenges in algorithm efficiency and data reduction. We concentrated our effort in three areas: 1. Facial animation on mobile phones, 2. Cooperative computer graphics in distributed environments and 3. Procedurally generated cities and buildings.

The aim of the thesis is to be a multidisciplinary publication that combines research results from fields of computer graphics and mobile networking. We focused on novel ways to utilize the properties of distributed mobile environments to perform graphical tasks and overcome various problems in distributed graphical applications caused by occasional unreliable mobile device network connections.

Chapter 1

Introduction

1.1 Goals of the Thesis

This thesis is focused on two main goals:

Our first goal is to implement graphical scenarios common on desktop computers yet thus far not possible on mobile devices due to their limited resources. We decided to especially focus on semantic-based methods for content reduction and procedural generation. These methods can reduce the amount of resources required by mobile devices and allow us to deliver new experiences to users. Semantic-based reduction can detect graphic content that is unimportant to the user and remove it from memory or network transfers. Such content might for example be outside the current view of the user or possibly replaced by other very similar content already present on the device. Procedural generation creates required content on demand rather than downloading it. The mobile device also saves memory by releasing certain cached content when it can be re-generated again as needed. We aim to propose graphical scenarios where procedural generation can save both bandwidth and memory resources.

The second goal of the thesis is to utilize wireless connection on mobile devices in new graphical scenarios. These devices can use traditional client-server or a modern client-cloud connection to offload computational-intensive tasks to powerful servers. Our goal is to decide how to distribute tasks between these two parts to save bandwidth, battery and other mobile device resources. Moreover certain mobile devices are capable of communicating directly (without routers or base stations) over short distances using Bluetooth or WiFi Direct technology. We attempt to utilize such communication in innovative graphical applications. We further propose how to deal with unreliable wireless connections of mobile devices in various graphical scenarios. Users may bring their smartphones to places

with no WiFi or cellular signal; Bluetooth connections may be interfered with or there will be no device within range. We attempt to design new graphical mobile applications that can work with only occasional connectivity.

1.2 Definition of Mobile Graphics

For our purposes we define mobile graphics as computer graphics displayed on a mobile device. However, there are several definitions for devices considered as mobile. These are the two most common:

1. A mobile device is a device that has a battery and can be fully used while walking.
2. A mobile device is a device that uses a mobile operating system. (see Figure1.1)

For our purposes the exact definition for mobile devices is not important and we take all devices as mobile as long as they confirm at least one of the above definitions.

Mobile devices usually have wireless connection to access remote servers and services. In many cases both data processing and storage are moved from the mobile device to powerful and centralized computing platforms located on servers or in clouds. These approaches may be also used in computer graphics and we discuss them in this thesis as well.

Figure 1.1: The most commonly used mobile operating systems - Google Android (left), Apple iOS (middle) and Microsoft Windows Phone (right). Less commonly used mobile operating systems are Symbian OS, BlackBerry OS, Palm OS and Windows Mobile

1.3 Hardware Components Used in Mobile Devices

Mobile devices have a range of specific hardware characteristics. In this section we discuss those that are important for computer graphics.

1.3.1 Mobile Touchscreen Displays

Usual display resolutions vary from 320×200 pixels (qVGA resolution) to 2560×1440 pixels (WQXGA resolution) for smartphones. Resolution of tablet devices can range up to 3200×1800 pixels (Lenovo Yoga 2 Pro). Mobile devices most commonly use LCD displays that consume constant power regardless of content displayed.

The second most frequent technology for mobile displays is AMOLED. The consumption of AMOLED displays depends upon the image displayed on the screen. An entirely white image on an AMOLED display consumes approximately three times more power that an LCD. However, for mostly black images the AMOLED displays are far more effective. This image-consumption dependency opens new challenges in mobile computer graphics. For example, Windows Phone operating system is designed for effective handling with an AMOLED display. The system extensively uses black color to reduce the total amount of display energy consumption.

Displays used in mobile devices may use various touch technologies:

- **Non-touch screen**
  Some mobile devices still lack a screen that responds to touch. This is particularly typical of devices with hardware keyboards. Most modern operating systems for mobile devices rely on touchscreen interfaces and cannot be used without the. A screen without touch support is typical for devices with Blackberry operating system, although the modern versions also support touch interfaces. Mobile devices with Windows Mobile and Android operating systems can also operate without a touchscreen ,although this option is rarely used.

- **Resistive touchscreens**
  The main disadvantage of resistive touchscreens is the support of only one touch point. The display responds to any pressure made by finger or passive stylus. Resistive touchscreens are now considered obsolete, although they are still used in the new Nintendo game tablet for console Wii U.

- **Capacitive touchscreen**

  Capacitive touchscreens respond to the touch of human fingers or conductive materials. Capacitive displays usually support multi-touch up to 10 fingers. Capacitive touchscreens on some devices do not respond to finger touch when the user wears gloves.

- **Active stylus**

  This touchscreen technology uses wireless communication between the screen and stylus to detect stylus position even when it is not touching the screen. Multiple other parameters can be also transmitted, such as stylus tip pressure, stylus tilt or state of other stylus buttons. Although this style of interaction has been used for many years in the domain of desktop and laptop computers it is quite new for tablets without a hardware keyboard. Active stylus can be used e.g. for Windows Surface Pro tablets or Samsung Galaxy Note tablets.

- **Other types of touch interaction**

  Rarely we do encounter new types of touchscreen interaction. PlayStation Vita uses a dual touch screen (a touch screen with a rear touchpad). This allows the user to control the application without obscuring the display. Some mobile devices use cameras for gesture recognition without touching the actual screen. There are also other systems based on infrared rays and detectors that work without actual touch. The disadvantage of the infrared approach is the raised edge of the display, due to the placement of infrared transmitters and receivers.

1.3.2 Connectivity

- **Wired connection**

  Wired data transfers once were used to synchronize mobile devices with desktop computers. In recent years synchronization is preferred using a cloud synchronization service accessible via a wireless network. However, wired synchronization is still used for transfers of large amounts of data, e.g. in multimedia collections. One indisputable benefit of using wired connections is that the phone can charge itself when connected, therefore some energy-demanding scenarios may prefer to run when external power is available. Most smartphones currently use a Micro-USB for charging and wired-data transfer.

- **Mobile internet connection**

  Mobile devices often have the capability of connection to wireless networks. There are several types of standards for wireless data transfer. Mobile operators usually use a pay-as-you-go business model, where users pay according to the amount of data transfer they use or a monthly-based data-transfer limit. Technologies used by mobile operators and their maximum transfer speeds are illustrated in the table below. Maximum speed of data transfer is important when planning a distributed scenarios for graphical applications.

Abbreviation	Full name	Max speed
GPRS (1G)	General Packet Radio Service	8 kB/s
EDGE (2G)	Enhanced Data rates for GSM Evolution	33 kB/s
UMTS (3G)	Universal Mobile Telecommunications System	48 kB/s
HSDPA (3.5G)	High-Speed Downlink Packet Access	225 kB/s
LTE (4G)	Long Term Evolution	12.5 MB/s
WiFi	WiFi	108 MB/s
Bluetooth	Bluetooth	3 MB/s

- **WiFi and WiMAX wireless technology standards**
  Usually neither WiFi nor WiMAX connection means a bottleneck for mobile applications. The bandwidth is in most cases sufficient for even the most demanding scenario such as video-streaming. However, problems may occur when multiple users want to utilize a single wireless router or when the speed of the wired connection of the router is not sufficient. WiFi connection is now supported by almost all kinds of mobile devices. On the other hand WiMAX, (an outdoor alternative for WiFi) connection is not currently available on many mobile devices, but we expect future expansion of supported devices.

- **Other connection possibilities**

  Some mobile device can use other types of wireless connections such as infrared, Bluetooth or NFC (Near Field Communication). Some mobile devices also use wireless streaming standards such as WiDi or Miracast.

1.3.3 Battery

Mobile devices require battery power. The processing power of integrated circuits increases according to Moore's law, roughly by 50% per year, although this is

certainly not the case of battery capacity. The energy capacity of batteries increases by approximately by 10% per year. This phenomenon is partially compensated by Gene's law. It states that the power usage of integrated circuits drops by half every 18 months. The effect of Gene's law allows building smaller and faster mobile devices.

1.4 Thesis Structure

The subsequent parts of the thesis are organized as follows:

In **Part I** we present our effort in the area of facial animation on mobile devices. We describe our supplementary semantic-based method for reduction of animated 3D polygonal models. Further, we present our framework for the easy creation of interactive, platform-independent voice-services with an animated 3D talking-head interface that uses our reduction method. Thise framework supports automated multi-modal interaction using speech and 3D graphics on mobile phones.

Chapter 2 explains the terms used in the area of facial animation on mobile devices and discusses applications of the technology.

In chapter 3 we present a novel supplementary semantic-based method for the reduction of animated 3D polygonal models. The method is mainly applicable in the animation of human faces and is based upon the semantic-based merger of visemes represented by key polygonal meshes. It is useful for devices with limited CPU and memory resources such as mobile phones or other embedded devices. Using this approach, we can reduce operation memory needs and time required to load the model from storage. We describe the algorithm for viseme merger and prove that our method is optimal for selected metrics. We also validate method performance on an example and compare it with the case when only traditional methods for 3D model reductions are used.

In chapter 4 we introduce our framework for the easy creation of interactive 3D talking-head applications on mobile phones. We address the difficulty of synchronizing audio stream to animation and discuss alternatives for distributed network control of animation and application logic. We also compare various possible architectures for talking-head applications and prove that our solution is most effective in the majority of cases.

Part II describes scalable cloud-based 3D environments that allow cooperation and interaction of multiple users in a single world. We discuss synchronization issues due to network and cloud-operation delays.

Chapter 5 describes our application for a cloud-controlled system for device-to-device content distribution. The aim of the application is to reduce the traffic load of cellular

networks by dynamically distributing content from the cloud to a subset of subscribed users and allow these users to spread the content with device-to-device communication.

In **Part III** we describe methods used for procedural city generation. We focus on recent and future approaches to achieve semantic-based reduction of generated cities in order to accomplish automatic level-of-detail and bring them to devices with limited resources such as embedded devices, PDA or mobile phones. We also discuss applications of city models on such devices.

Chapter 6 describes recent approaches in the area of procedural city modeling. We further describe standard city modeling workflow and current open problems related to automatically generating building models in multiple level-of-detail and generating infinite city models in real-time.

In chapter 7 we present our stateless generation algorithm that creates infinite on-demand generated 3D virtual worlds in distributed environments. The algorithm can be useful in multiuser virtual worlds with mobile clients using independent geometry generators. The algorithm ensures that overlapping areas will contain the same geometry for all clients.

Conclusions of our research are presented in chapter 8.

Part I

Rendering of Facial Models

This part is based on our work published in [A.2], [A.5], [A.10], [A.11], [A.12] and [A.13].

Current smartphones and pocket computers have rather limited RAM memory. A large part of this memory is occupied by the operating system (OS) itself or by OS extensions like HTC TouchFLO or Samsung TouchWiz. Early pocket computers had only 16 or 32 MB of operation memory, but the OS was stored in read-only memory rather than in RAM. Lack of memory is a bottleneck for animations computed by interpolation of polygonal meshes, because it requires many possibly large polygonal meshes loaded in the memory.

To achieve the lowest memory requirements, we decided to create an algorithm that allows us to reduce both the amount of polygons in the mesh and the number of key meshes. We propose a dissimilarity metric to detect similar keyframe-animation meshes, as well as a technique to merge them. We also prove our merging technique is optimal for the given dissimilarity metric.

Chapter 2

Facial Animation on Mobile Devices

2.1 Introduction to Facial Animation

The rapid proliferation of mobile devices over the past decade and their enormous improvements in computing power and display quality opens new possibilities in using 3D representations for complementing voice-based user interaction. Their rendering power allows creation of new user interfaces that combine 3D graphics with speech recognition and synthesis. Likewise, powerful speech-recognition and synthesis tools are becoming widely available to mobile clients or readily accessible over the network, using standardized protocols and APIs.

An interface with a 3-dimensional talking head on a mobile phone display represents a promising alternative to the traditional menu/windows/icons interface for sophisticated applications, as well as a more complete and natural communication alternative to purely voice- or tone-based interaction. Such interfaces have proven to often be useful as a virtual news reader [7], weather forecast [60], healthcare communication assistant [52], and blog enhancement [61]. The talking-head interfaces can be especially useful in developing regions where people often cannot read and write.

2.1.1 Applications of Facial Animation

3D user interfaces are a general trend across multiple disciplines [17], due to their natural interaction aspect and the increasing availability of relevant technology. In the domain of desktop computing, with large displays and multimedia support, the use of multi-modal interaction and 3D virtual characters has been on the rise. Virtual characters improve telepresence (the notion of customer and seller sharing the same space) in e-commerce [95] as well as interaction with technology for elderly people [82]. Learning exercises with

virtual characters [116] have shown that audio components improve their perception and that 3D virtual characters are far better perceived than 2D ones. Much effort has also been concentrated on building multi-modal mobile interaction platforms [29].

Research into Embodied Conversational Agents (ECA), agents with a human shape using verbal and non-verbal communications [31], shows that people prefer human-like agents over caricatures, abstract shapes or animals and, moreover, agents with a similar personality to their own [31, 80].

2.1.2 Applications of Voice Interfaces

Natural interaction with the resources of the global network (especially using voice) is a growing field of interest. Recent works, for example, have developed the idea of the World Wide Telecom Web (WWTW) [4, 5, 59], a voice-driven ecosystem parallel to the existing WWW. It consists of interconnected voice-driven applications hosted in the network [59], a *Voice Browser* providing access to the many voice sites [4] and the Hyperspeech Transfer Protocol (HSTP) [5] allowing their seamless interconnection. Developing regions with a large proliferation of phones but little Internet literacy are set to benefit.

Similarly, mobile platforms will benefit from improved interaction. For example, mobile Web browsing has been shown to be less convenient than desktop browsing [101]. Augmenting the interaction with voice and graphics assistance ought to improve it. Conversely, pure voice-response systems have been shown to benefit from augmenting with a visual interface [126]. This motivates the addition more modalities into the mobile-user interactions with Web.

Research in assistive technologies has focused on Web interaction by voice and its applicability for the handicapped or elderly. For example, the HearSay audio Web browser [98, 111] allows for automatic creation of voice applications from web documents. An even larger group of handicapped may be reached if more modalities are used for interaction, allowing the use of animations or sign-language.

2.1.3 Head Models for Facial Animation

Detailed 3D-face rendering has so far avoided the domain of mobile clients, due to limited computing capacity, display quality and battery lifetime. Previous attempts to render an avatar face on a mobile client continue to use non-photorealistic rendering (NPR), such as the cartoon shading [22]. The platform in [22] also has ambitions for strong interactivity, allowing for visual interaction based on video capture and server-based face-expression

recognition. However, the character is not automated, but merely conveying the visual expression of the person at the other end of the communication channel.

A more advanced physics-based model, dedicated for powerful desktop computers or offline rendering [6], have also been presented. This model is relying on co-articulation – the coloring of a speech segment by surrounding segments and a distributed model [91] (based on phoneme timestamps) for synchronizing facial animations with speech.

2.1.4 Existing Scripting Languages for Facial Animation

Several languages convenient for talking-head scripting are available. We exploit the SMIL-Agent (Synchronized Multichannel Integration Language for Synthetic Agents) [13] scripting language, based on XML. Related languages developed for talking head scripting are AML (Avatar Markup Language) [58] and ECAF (Authoring Language for Embodied Conversational Agents) [60].

An open modular facial-animation system has been described in [118]. Commercial systems such as FaceGen [104] can be used to create face meshes and the Xface [11] represents an open toolkit for facial animations. We take inspiration from these tools, targeted for PC platforms and extend them with network connection functionality, taking the features of mobile clients and their power-consumption limitations into consideration.

2.2 Face Animation Principles

2.2.1 Phonemes and Visemes

When using face animation in talking-head applications, we must consider both visual and audio effects. They are described as visemes and phonemes. A phoneme is an element of spoken language, as a letter is an element of written language. A viseme is an element of facial animation. It describes the particular facial position when pronouncing a phoneme. Usually one phoneme corresponds to one viseme, but occasionally multiple phonemes share the same viseme. This occurs when the facial position of two or more phonemes differ only by the position of non-displayed body parts such as vocal cords or a tongue.

The frequencies of occurrence of phonemes and visemes depend on spoken language and there are also differences e.g. between frequencies in British and American English. Both British and American English have 40 different phonemes.

For our reduction algorithm described in the next chapter we must know the frequencies of phonemes and visemes. The frequencies of phonemes can be determined converting

a long text (at least several pages) using a phonetic transcription software and then by counting the phoneme frequencies in the transcribed text. This process is usually part of text-to-speech-engine pre-processing of text input for voice synthesis. There is also a free transcription engine available together with typical frequencies of American English phonemes [34]. Knowing the frequencies of phonemes, one can determine the frequencies of visemes using the phoneme-to-viseme mapping function. It usually occurs that multiple phonemes map onto one viseme.

For our experiments we use the FaceGen facial editor [104] to generate human head visemes. This editor generates 16 different visemes.

2.2.2 MPEG-4 Animation

The most widely accepted standard for human face animation is the ISO standard MPEG-4 released by the Moving Pictures Experts Group in 1999 [46,47]. In this standard 84 feature points (FPs) are specified on the human face (see figure 2.1). The facial animation is controlled by 68 parameters called Facial Animation Parameters (FAPs).

The MPEG-4 standard allows two ways of facial animation. The first manipulates feature points individually and can achieve various ranges of facial expression. The second is based on interpolating between two keyframe meshes. This interpolation can be done either linearly (see equation 2.1) or with the cubic interpolation function (see equation 2.2).

$$m(t) = (1-t)\cdot m(0) + t\cdot m(1);\ t \in <0,1> \tag{2.1}$$

$$m(t) = (1-3t^2+2t^3)\cdot m(0) + (3t^2-2t^3)\cdot m(1);\ t \in <0,1> \tag{2.2}$$

In our research we focus on keyframe facial animation, using linear interpolation because it is faster. This approach is less CPU intensive and the visual results of this animation are sufficient for mobile phones and embedded devices. The advantage of the keyframe-morphing system is simplicity and speed of implementation, but it requires a lot of space for model storage and much work to prepare the keyframe meshes.

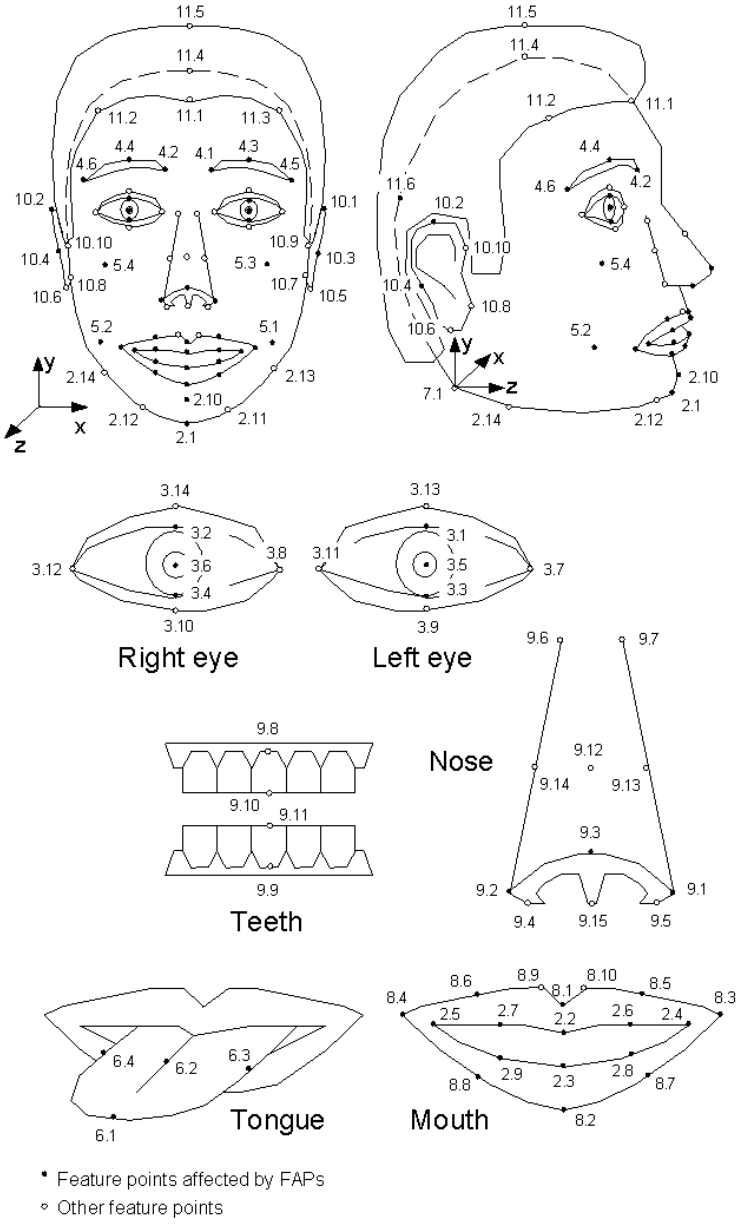

Figure 2.1: Feature points (FP) defined in MPEG-4 facial animation standard [47]

Chapter 3

Semantic Reduction of Face Meshes

This chapter describes our novel algorithm that allows reduction in both the amount of polygons in a mesh of a face model and the number of keyframe-animation meshes (see figure 3.1).

3.1 Existing Reduction Methods

Traditional methods for the reduction of polygonal models are based on reducing the number of polygons. These methods are sometimes not sufficient to reduce the size of a head model. Therefore we focused on methods for reducing the number of polygonal meshes. Using both the reduction of the number of polygons and the reduction of the number of polygonal meshes, we are able to save far more operation memory. Traditional methods for polygonal reduction are sufficiently covered by Kurs et al. [57] and Pasman et al. [87] – see Figure 3.2 for samples of results. Specific aspects concerning geometric rendering and model reduction on mobile phones and embedded devices were presented by Pulli et al. [94].

An interesting way to speed up morphing animation on embedded devices was proposed by Berner [14]. It is based on optimization strategies that omit less important polygonal meshes during animation.

In our research we aim to develop software compatible with the Xface animation framework developed by Balci et al. [10, 12] that is open-source and widely used in academia. There are also more advanced animation frameworks that use a skeleton-muscle animation model published by Sifakis et al. [102] instead of MPEG-4 standard. The best known of them is Greta [36, 81, 88]. A method of anatomical musculature modeling to achieve realistic and real-time figure animation was proposed by Zuo Li et al. [66].

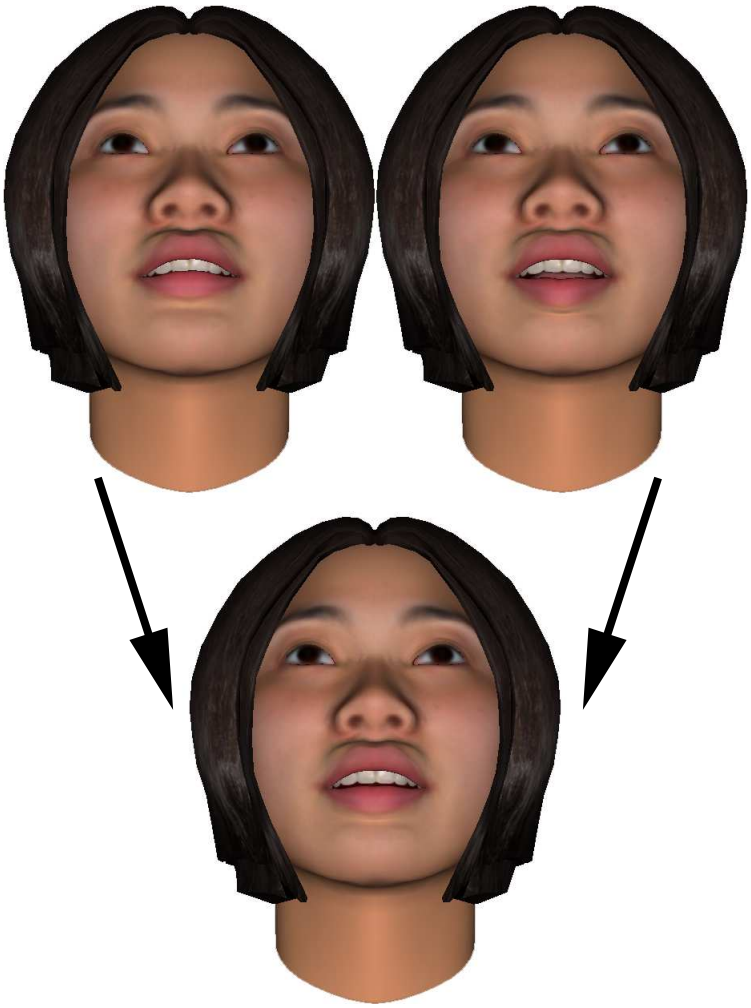

Figure 3.1: **Mergeing of keyframes meshes of faces**: A talking head keyframe mesh articulating the phoneme ”f” (left) is similar to a keyframe mesh articulating the phoneme ”th” (right). Our algorithm detects such similarity and replaces both meshes with one merged mesh (down).

However none of the above works focus on reducing the number of visemes (as we do).

3.2 Definitions

Polygonal mesh

For the purposes of this thesis, the polygonal mesh is a triplet (V, E, P) of vertices V, edges E, and polygons P. To avoid rendering problems with general polygons after geometric transformations, we triangulate all polygons in advance.

Fully triangulated meshes allow us to use a specific metric for mesh comparison (see

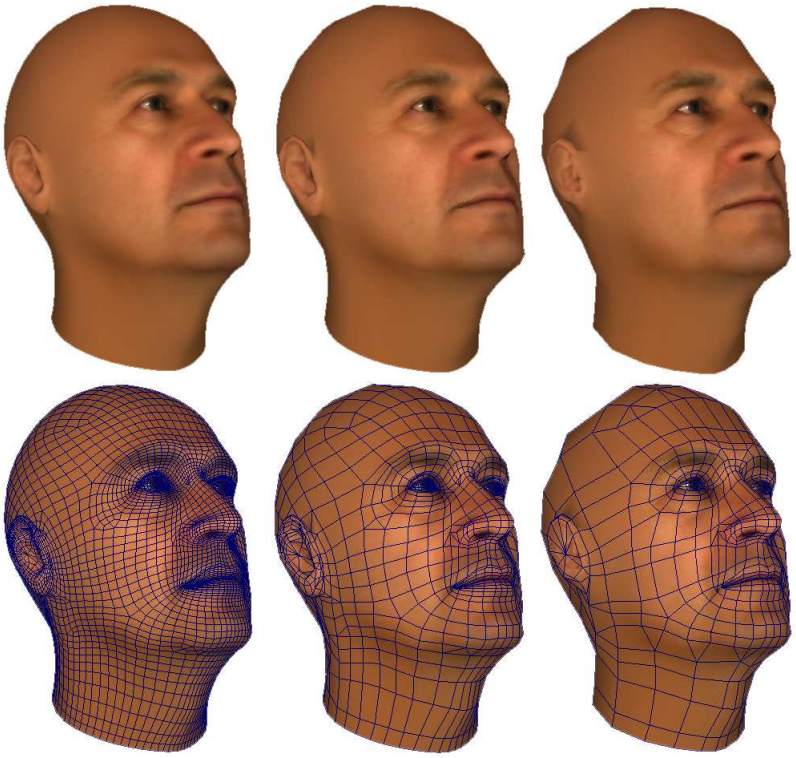

Figure 3.2: Basic model reduction can be done with reducing the number of polygons. Applications on powerful desktop PCs use high-polygon meshes (left). For mobile phones and embedded systems we use low-polygon meshes (right). If the model size is not sufficiently reduced, it is possible to reduce the number of visemes (key meshes) using our reduction algorithm.

section 3.2.1). They also fit very well into commonly used graphics libraries for mobile phones and embedded devices like OpenGL ES (OpenGL for Embedded Systems) [56] which are optimized only for processing triangles.

Interpolable set of meshes

We call a pair of polygon meshes interpolable if they differ only in coordinates of their vertices. Interpolable meshes have the same topology and the same number of vertices, edges and polygons. There must also be given a bijection function that matches the corresponding vertices/edges/polygons. We call a set of meshes interpolable if every pair of meshes from the set is interpolable. A collection of facial animation visemes is a typical example of an interpolable set of meshes. A set of keyframe-animation meshes also forms an interpolable set of meshes.

3.2.1 Polygonal-mesh Dissimilarity

We define the polygonal-model dissimilarity as a metric (distance function) ρ for two interpolable meshes.

$$\rho(A, B) := \sum_{k=1}^{\|V\|} w(v_k)\|v_{A,k} - v_{B,k}\|^2 \tag{3.1}$$

where
A and B are the polygonal meshes.
$w(v)$ is the weight of the vertex v. It represents the importance of the vertex in the model. The author of the model can set higher weights for vertices important for human perception.

For models with unspecified weights, we have considered two general metrics:

$$\rho_1(A, B) := \sum_{k=1}^{\|V\|} \|v_{A,k} - v_{B,k}\|^2 \tag{3.2}$$

$$\rho_2(A, B) := \sum_{k=1}^{\|V\|} S(v_{N,k}) \|v_{A,k} - v_{B,k}\|^2 \tag{3.3}$$

where
$S(v_{N,k})$ is a sum of surfaces of triangles incident with vertex $v_{N,k}$. Since the triangle surface may differ for individual visemes, we work with polygon surfaces in the neutral-expression mesh of the model $N = (V_N, E_N, P_N)$.

The first metric assumes that more important areas of the model are modeled in higher detail and thus contain more vertices. The weight of a model area (e.g. cheek, lip, chin) is estimated by the number of its vertices. The more vertices a certain model area has the more time it affects the polygonal model dissimilarity due to the sum $\sum_{k=1}^{\|V\|}$.

The second metric supposes each model area surface is equally important for the animation. If we use this metric it is necessary to first split all polygons into triangles as mentioned in section 3.3. We have proven that both metrics give the same results if applied in our reduction algorithm. Thus the actual implementation can utilize the first and simpler metric.

3.2.2 Dissimilarity for Sets of Polygonal Meshes

Let $\mathbb{A} = \{A_1, A_2, \ldots, A_n\}, \mathbb{B} = \{B_1, B_2, \ldots, B_m\}$ are two sets of polygonal meshes that represent visemes of the same face model. Let $f(A_1), f(A_2), \ldots, f(A_n)$ are frequencies of visemes in $\mathbb{A}$. If we have a dissimilarity metric for polygonal meshes $\rho(A, B)$, we can

define dissimilarity for two sets of polygonal meshes $\rho_f(\mathbb{A}, \mathbb{B})$ as:

$$\rho_f(\mathbb{A}, \mathbb{B}) = \sum_{i=1}^{n} f(A_i) \min_{j=1...m} \rho(A_i, B_j) \tag{3.4}$$

The formula contains the sum of distances from each mesh from $\mathbb{A}$ to its most similar mesh in $\mathbb{B}$. Note that the dissimilarity function for sets of polygonal meshes is not a metric because it is not symmetrical.

3.2.3 Reduction Problem Definition

We describe an algorithm for the following problem:
Input:
Set of polygonal meshes $\mathbb{A} = \{A_1, A_2, \ldots, A_n\}$. These meshes represent visemes of a human face model that have frequencies $f(A_1), f(A_2), \ldots, f(A_n)$. An integer number m; $m < n$
Task:
Find a set of new polygonal meshes with m elements $\mathbb{B} = \{B_1, B_2, \ldots, B_m\}$ that is the most similar to $\mathbb{A}$. ($\rho_f(\mathbb{A}, \mathbb{B})$ is minimal for all such sets of polygonal meshes)

3.3 Finding the Optimal Solution

The solution for the problem is described in two steps: Firstly, we describe how to solve the extreme case when $m = \|\mathbb{B}\| = 1$. Then we describe the solution for an arbitrary value of $\|\mathbb{B}\|$.

Case $\|\mathbb{B}\| = m = 1$

We must find such a set of polygonal meshes $\mathbb{B} = (B)$ with one element for which the expression in equation (3.4) is minimal.

$$\mathbb{B} = \underset{\mathbb{B}\,;\,\|\mathbb{B}\|=1}{\arg\min}(\rho_f(\mathbb{A}, \mathbb{B})) \tag{3.5}$$

We use the definition of the dissimilarity for sets (see equation (3.4)):

$$\mathbb{B} = \underset{\mathbb{B}\,;\,\|\mathbb{B}\|=1}{\arg\min}(\sum_{i=1}^{n} f(A_i) \min_{j=1...m} \rho(A_i, B_j)) \tag{3.6}$$

Because $m = 1$, we can leave out the second minimum.

$$B = \arg\min_{B}(\sum_{i=1}^{n} f(A_i)\rho(A_i, B)) \tag{3.7}$$

Now we use the definition of mesh dissimilarity metric (see equation (3.1)).

$$B = \arg\min_{B}(\sum_{i=1}^{n} f(A_i) \sum_{k=1}^{\|V\|} w(v_k)\|v_{A_i,k} - v_{B,k}\|^2) \tag{3.8}$$

We swap the summations.

$$B = \arg\min_{B}(\sum_{k=1}^{\|V\|} \sum_{i=1}^{n} f(A_i)w(v_k)\|v_{A_i,k} - v_{B,k}\|^2) \tag{3.9}$$

Since the vertices of mesh B are mutually independent, we can calculate each of them individually.

$$V_{B,k} = \arg\min_{V_{B,k}}(\sum_{i=1}^{n} f(A_i)w(v_k)\|v_{A_i,k} - v_{B,k}\|^2) \tag{3.10}$$

The vertex weight $w(v_k)$ remains constant for an individual vertex. Thus it does not affect the $\arg\min$ expression. It can be left out.

$$V_{B,k} = \arg\min_{V_{B,k}}(\sum_{i=1}^{n} f(A_i)\|v_{A_i,k} - v_{B,k}\|^2) \tag{3.11}$$

We use the definition of the Euclidian distance.

$$v_{A_i,k} = [x_{A_i,k}, y_{A_i,k}, z_{A_i,k}] \tag{3.12}$$

$$v_{B,k} = [x_{B,k}, y_{B,k}, z_{B,k}] \tag{3.13}$$

$$\begin{aligned} V_{B,k} \quad = \quad & \arg\min_{[x_{B,k},y_{B,k},z_{B,k}]} \sum_{i=1}^{n} f(A_i)(x_{A_i,k} - x_{B,k})^2 + \\ & + f(A_i)(y_{A_i,k} - y_{B,k})^2 + f(A_i)(z_{A_i,k} - z_{B,k})^2 \end{aligned} \tag{3.14}$$

We can determine individual coordinates separately, because they are independent of each

other. Let us consider the x-coordinate only:

$$x_{B,k} = \arg\min_{x_{B,k}} \sum_{i=1}^{n} f(A_i)(x_{A_i,k} - x_{B,k})^2 \tag{3.15}$$

We expand the expression.

$$x_{B,k} = \arg\min_{x_{B,k}} \sum_{i=1}^{n} f(A_i)(x_{A_i,k}^2 - 2x_{A_i,k}x_{B,k} + x_{B,k}^2) \tag{3.16}$$

In order to find the minimum, we find where the derivation is equal to 0.

$$0 = \frac{\partial}{\partial x_{B,k}} \sum_{i=1}^{n} f(A_i)(x_{A_i,k}^2 - 2x_{A_i,k}x_{B,k} + x_{B,k}^2) \tag{3.17}$$

After the derivation we arrive at:

$$0 = \sum_{i=1}^{n} f(A_i)(-2x_{A_i,k} + 2x_{B,k}) \tag{3.18}$$

The second derivation is equal to $2\sum_{i=1}^{n} f(A_i)$. This is greater than 0 because all of the frequencies are positive. Thus this is a minimum. We express the $x_{B,k}$.

$$x_{B,k} = \frac{\sum_{i=1}^{n} f(A_i)x_{A_i,k}}{\sum_{i=1}^{n} f(A_i)} \tag{3.19}$$

We express the vertex $v_{B,k}$:

$$v_{B,k} = \frac{\sum_{i=1}^{n} f(A_i)v_{A_i,k}}{\sum_{i=1}^{n} f(A_i)} \tag{3.20}$$

We finally express the mesh B:

$$B = \frac{\sum_{i=1}^{n} f(A_i)A_i}{\sum_{i=1}^{n} f(A_i)} \tag{3.21}$$

Case $\|\mathbb{B}\| = m > 1$

We must find such a set of polygonal meshes $\mathbb{B} = (B_1, B_2, \ldots, B_m)$ with m elements for which the expression in formula 3.4 is minimal.

$$\mathbb{B} = \arg\min_{\mathbb{B}\,;\,\|\mathbb{B}\|=m} (\rho_f(\mathbb{A}, \mathbb{B})) \tag{3.22}$$

We use a dynamic programming approach:
Let $minDis[\mathbb{T}, p]$ be an array of real numbers indexed by a subset $\mathbb{T} \subset \mathbb{A}$ and an integer $p \in \{1 \dots m\}$ defined as:

$$minDis[\mathbb{T}, p] := \min_{\mathbb{U}\,;\,\|\mathbb{U}\|=p} (\rho_f(\mathbb{T}, \mathbb{U})) \tag{3.23}$$

This array represents the distance for all subsets of $\mathbb{A}$ to its optimal reductions of size p. If we are able to fill the array, we can find the answer to our problem in the field $minDis[\mathbb{A}, m]$. We describe an algorithm to fill the array $minDis[\mathbb{T}, p]$ with values. For $p = 1$ we can use the equation (3.21).

$$minDis[\mathbb{T}, 1] = \rho_f(\mathbb{T}, \{\frac{\sum_{i=1}^{n} f(T_i)T_i}{\sum_{i=1}^{n} f(T_i)}\}) \tag{3.24}$$

Now we can increase the value of p step-by-step and compute the values of the remaining fields of the array $minDis$. We try to find a subset $\mathbb{V} \subset \mathbb{T}$ that is reduced to a single mesh during the optimal reduction. The reduction is optimal if the sum of reduction of $\mathbb{V}$ to one mesh and reduction of $\mathbb{T} \setminus \mathbb{V}$ to $p - 1$ meshes is minimal.

$$minDis[\mathbb{T}, p] = \min_{\mathbb{V} \subset \mathbb{T}} (minDis[\mathbb{V}, 1] + minDis[\mathbb{T} \setminus \mathbb{V}, p - 1]) \tag{3.25}$$

Using the algorithm above we can compute the dissimilarity during the optimal reduction. We can easily find the set $\mathbb{B}$ itself by making notes on the performed reductions (found sets $\mathbb{V}$) during the algorithm.

The time complexity of the algorithm is $O(n2^n\|V\| + 4^n m)$. The spacial complexity of the algorithm is $O(n\|V\| + 2^n m)$. The algorithm is exponential to n. This is not a principal drawback because the values of n and m are small (we usually use a modeling software that exports each model as a set of meshes with 16 visemes) and we use this reduction only once as a pre-processing for each set of meshes.

Algorithm 1 Algorithm for optimal mesh reduction

input A
input $f(A_1), f(A_2) \dots f(A_n)$
input m
for $\mathbb{T} \subset \mathbb{A}$ **do**
    $minDis[\mathbb{T}, 1] := \rho_f \left(\mathbb{T}, \frac{\sum_{i=1}^{n} f(T_i) T_i}{\sum_{i=1}^{n} f(T_i)} \right)$
end for
for $p := 2$ **to** m **do**
    for $\mathbb{T} \subset \mathbb{A}$ **do**
        $currentMinDistance := \infty$
        for $\mathbb{V} \subset \mathbb{T}$ **do**
            $distance := minDis[\mathbb{V}, 1] + minDis[\mathbb{T} \setminus \mathbb{V}, p-1]$
            if $distance < currentMinDistance$ **then**
                $currentMinDistance := distance$
            end if
        end for
        $minDis[\mathbb{T}, p] := currentMinDistance$
    end for
end for
output $minDis[\mathbb{A}, m]$

3.4 Implementation

We have implemented the algorithm in Java. For our measurement we used a computer with Intel Core Duo processor T8300 2.4GHz with 2 GB of RAM. (Our implementation is single thread only.) We measured the time needed to reduce 16 visemes to 10 visemes. Each of these visemes was represented by a polygonal mesh with 3000 triangles. Initial reductions for the case $p = 1$ took 2 minutes and 43 seconds. Dynamic programming reductions for the case $p > 1$ took 2 minutes and 23 seconds. Input/output operations took 12 seconds. The total time was 5 minutes and 18 seconds.

We use VRML (Virtual Reality Markup Language) as our input and output format for polygonal meshes. The output from our application is compatible with XfaceEd face editor proposed by Balci [12].

3.5 Performance Validation

To validate the performance of the reduction method we compared animation of a head before and after reduction on several head meshes exported from a head modeling software

FaceGen [104]. We used a textured head model in a resolution that has approximately 3000 triangles (the exact number differs based on the model properties used - e.g. male or female). The exported unreduced set always contained 16 visemes. For our measurements we used Windows Mobile phone HTC Touch Pro with OpenGL ES [56] support. From our previous experience we know that we have to reduce the number of visemes to at least 10 to ensure all the geometry will fit into the device memory. Therefore we have used reduction to 10 visemes.

Our mobile-rendering application with the unreduced number of meshes required 18 seconds for startup, while the same application with the reduced number of meshes required only 8 seconds for startup. The size of input file with geometry of meshes was 960 kB for the unreduced and 609 kB for the reduced version. The speed of the model animation was 5.4 FPS for the unreduced and 12.2 FPS for the reduced version. The unreduced version was slowed by memory swapping. The animation of the reduced version appeared much smoother thanks to higher achieved framerate.

Although our reduction method primarily focuses on the head animation it is sufficiently general for use in other animation techniques using polygonal mesh interpolation (e.g. human body, animals).

Chapter 4

Framework for Creating 3D Head Applications

4.1 Brief Framework Description and Related Work

We designed and developed a framework for the easy creation of interactive, platform-independent voice-services with an animated 3D talking-head interface, on mobile phones. The framework supports automated multi-modal interaction using speech and 3D graphics.

We address the difficulty of synchronizing the audio stream to the animation and discuss alternatives for distributed network control of the animation and application logic. We document the ability of modern mobile devices to handle such applications and show that the power consumption trade-off of rendering on the mobile phone versus streaming from the server favors the phone.

The tools presented will help empower other developers and researchers in future research and usability studies in the area of mobile talking-head applications. These may be used, for example, in entertainment, commerce, health care or education.

By providing a general tool to create interactive talking-head applications on mobile platforms, we aim to spark future research in this area. It may open up space for many useful applications, such as interactive mobile virtual assistants, coaches or customer-care, e-government platforms, interactive assistants for the handicapped, elderly or illiterate, as well as 3D gaming or navigation, quiz competitions or education [116]. It may be used for secure authentication, for enriching communication with emotional aspects or for customizing the communicating-partner's appearance.

3D talking-heads have their disadvantages as well - consuming a lot of resources and not being appropriate for all types of information exchange (such as complex lists or maps).

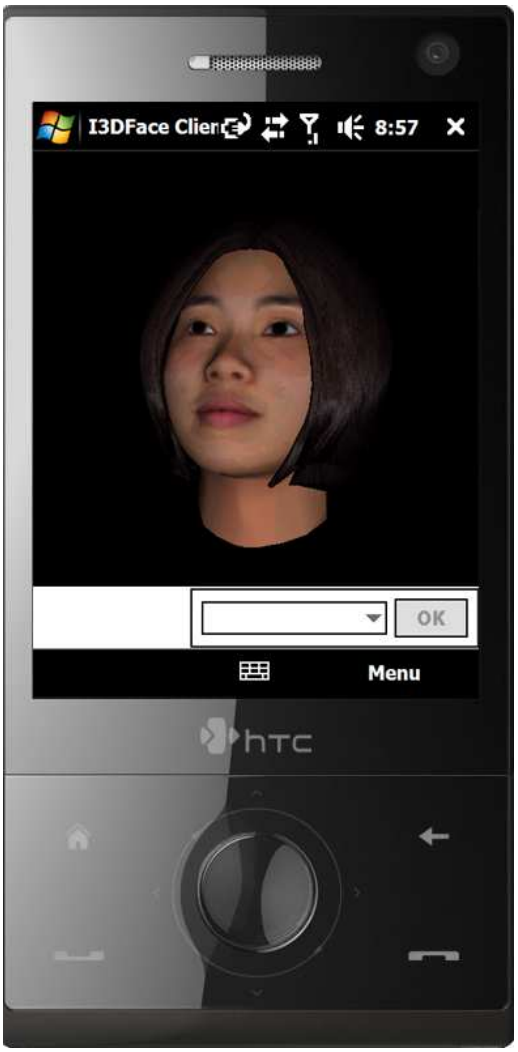

Figure 4.1: Talking-head application on a Windows Mobile 6.1 device (HTC Touch Pro). It is capable of articulating speech phonemes and showing facial expressions (anger, disgust, fear, sadness, smile, surprise).

The first aspect should take care of itself by the evolution of computing power, the second by adding further modalities to the interactive environment.

Previous existing mobile frameworks for easy application creation [49, 83, 85] were restricted to a particular mobile platform, yet there currently exist many mobile operating systems. Our proposed framework is not only platform independent, but also compatible with desktop facial-modeling tools.

4.2 Distributed Design Analysis

During the design process of our framework we considered several possible architectures for talking-head-enhanced applications. For a natural conversation between the (real) user and the (virtual) head we need components for 3D rendering, speech recognition, speech synthesis, and application logic. Each of these components can either reside on the client or server side. This section discusses possible architecture alternatives (see also Figures 4.2, 4.3 and 4.4).

4.2.1 Speech Synthesis

Speech can be synthesized either on the mobile device or a remote server. In the past, components for speech synthesis (also called Text-to-Speech engines) on mobile devices had somewhat lower quality than components for synthesis on desktop/server PCs, which possess more resources. However, the computational power and available memory of present mobile devices allows generating voice output with a quality that satisfies the needs for computer-human dialogue. So the impact in quality is virtually unrecognizable.

Synchronizing speech and face animation (lip movement) is a challenging task. We address the synchronization problem by using phoneme/viseme timestamps [91] (for details of the complete solution see Section 4.5). For this type of synchronization, speech synthesis and animation components must be co-located together. This is why we only support speech synthesis on the client. Nevertheless, as discussed in Section 4.3, client-side synthesis is more energy-efficient and should therefore be preferred over the server-side variant.

Synchronizing voice (speech) with animation (lip movement) has also been addressed in previous work, but mainly on desktop platforms. The BEAT animation toolkit [19] (based on language tagging) allows animators to input text to be spoken by an animated head and to obtain synchronized nonverbal behaviors and synthesized speech that can be input to a variety of animation systems. The DECface toolkit [119] focuses on correctly synchronizing synthesized speech with lip animation of virtual characters.

4.2.2 Speech Recognition

Speech recognition is significantly more CPU- and memory-intensive than speech synthesis. Suitable mobile speech-recognition solutions are available for scenarios where the set of recognized words is greatly limited (e.g. yes/no answers, one-of-N options or voice dial). Without such constraints (e.g. dictating an arbitrary letter), available mobile solutions are

quite error-prone. For speech recognition it is better in such cases to send the recorded voice to a remote server.

Unlike in the case of speech synthesis, our framework supports both server- and client-side speech recognition. However, client-side speech recognition is limited to very small dictionaries (about 50 words) with a simple acoustic and language model.

4.2.3 Graphics Rendering and Streaming

Visual application content can be rendered either on a mobile phone or a remote server followed by video-streaming to the phone. The second approach can be easily implemented as platform-independent because there is little code on the client side. Yet it has also multiple disadvantages.

Video streaming requires a lot of bandwidth that is often limited in mobile networks. Such architecture moves most components to the server side (see Fig. 4.2). The server renders video, synthesizes and both are then streamed over the network to the client. The entire application logic resides on the server side.

We attempted the video-streaming approach and our experiments show that latency of up to 400 ms, caused by video compression and network latency, may occur between the user input and response from the server. Such latency may make voice interface unpleasant, especially if the user expects an immediate response (e.g. using buttons to move a camera within a virtual world). Video-streaming on mobile phones is usually also more power-demanding.

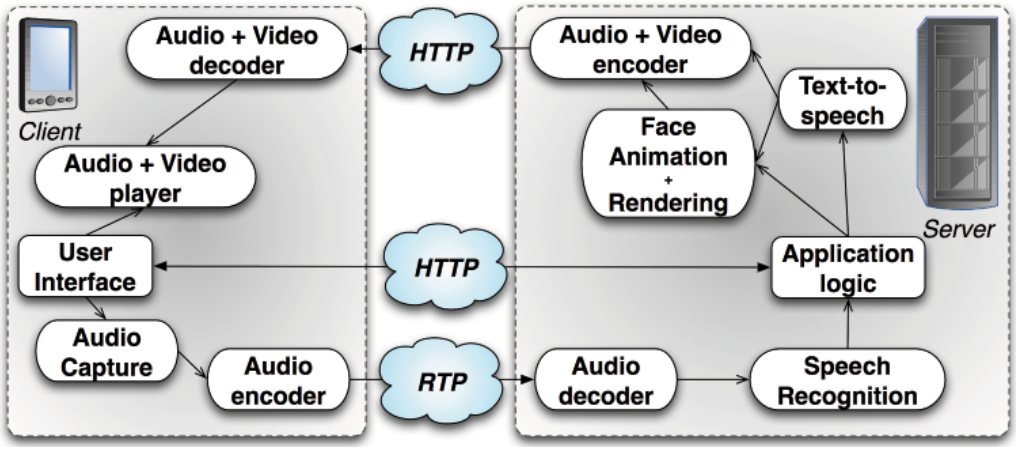

Figure 4.2: Video-streaming architecture is convenient for less powerful mobile phones with fast Internet connections, because it delegates most of the application work to a remote server. It can easily be implemented as platform-independent. We did not include such architecture in our framework, because it is not energy efficient.

Client-side graphics rendering is less power-demanding, however, it is far more challenging in its implementation as platform-independent and with the limited resources of a mobile systems. Different mobile phone platforms and devices have different rendering capabilities with different APIs. In our framework we use OpenGL ES [56] as the most common and platform-independent mobile rendering API. For head/face rendering we use models generated from FaceGen [104] editor with applied polygon reduction [40, 69, 99] and viseme reduction techniques as described in the previous chapter to reduce the model complexity.

4.2.4 Connection Requirements

According to our experiments, at least a 100 kbps connection throughput is needed for video streaming; otherwise the video quality is not acceptable for a user on a mobile client screen with a resolution 320x240. For audio streaming architectures (see Fig. 4.3), 12 kbps data connection is sufficient. The usual throughput on connections for mobile phones is: GPRS 40 kbps, EDGE 100 kbps, UMTS 300 kbps, Wi-Fi on mobiles 600 kbps. While audio streaming works over all of the above, video streaming requires a higher-bandwidth connection.

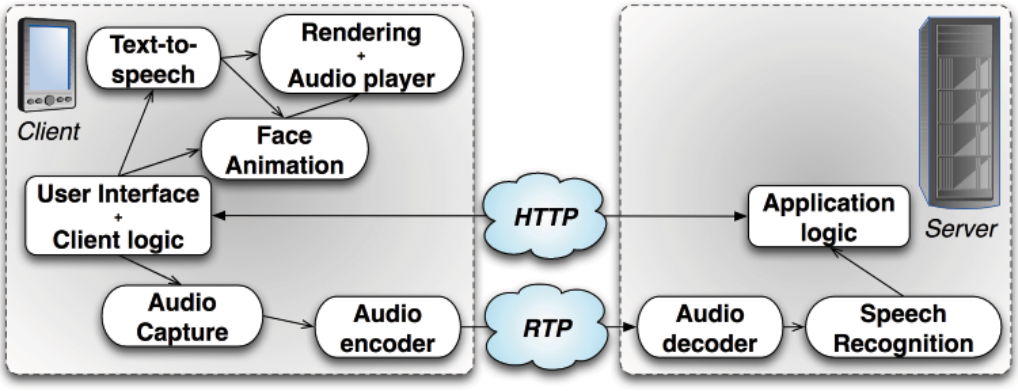

Figure 4.3: Client-server configuration using the server for application-logic processing and speech recognition. Results of the recognition process are directly provided to the application-logic module. The client side is used for text-to-speech processing, face animation and their synchronization. This architecture is supported by our framework.

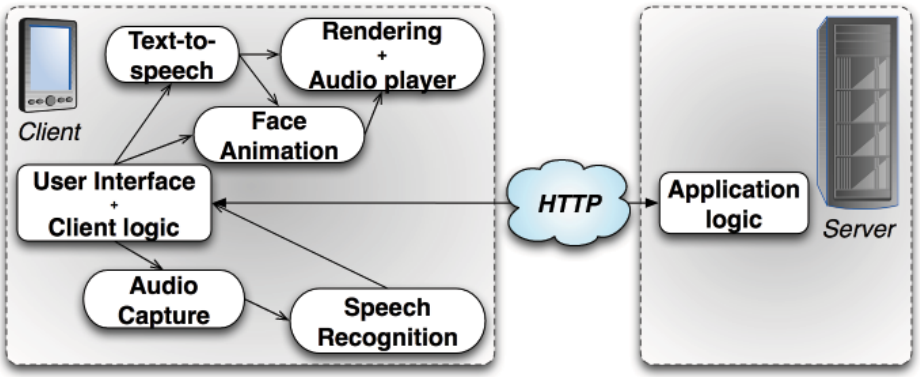

Figure 4.4: Client-server configuration using only the server side for application-logic processing. Our framework supports this type of configuration. It is suitable only for mobile devices with high computational power. This configuration is also convenient in situations where only low bandwidth is available.

4.3 Performance Measurements

4.3.1 Graphics Benchmarks

We performed several benchmark tests to validate 3D rendering performance and power consumption. For CPU utilization measurement we used the acbTaskMan utility [2]. All measurements and tests were performed on the HTC Touch Pro mobile device with Qualcomm 528 MHz processor and Windows Mobile 6.1 operating system. Qualcomm chipsets are the most common in current Windows Mobile phones. For demonstration and testing we developed an OpenGL ES rendering application called GLESBenchmark (see Fig. 4.5), inspired by [53], which renders a 3D head in real-time. Selected performance test results are summarized in Table 4.1.

We conclude that the phone is able to render up to 8000 triangles illuminated by one directional light at 15 frames per second, but the speed drops considerably when using a point light. Surprisingly, the rendering speed does not depend on the choice of shading method (flat or smooth shading).

According to GLBenchmark [53], some other phones (iPhone, Symbian phones) have no difficulty with rendering 3D objects illuminated by point light (the rendering speed is nearly the same as in the case of the directional light). Textures only affect rendering performance minimally.

We used a 512x512 pixel texture in our experiments. Maximum texture size in OpenGL ES is limited to 1024x1024 pixels or less on most mobile platforms.

Figure 4.5: Snapshots of the GLESBenchmark application created. The head is animated during performance measurements.

4.3.2 Power Consumption

We made estimates and rough measurements of power consumption for each of the architectures discussed. During tests, the Wi-Fi module with audio streaming was on, the display backlight was set to its minimum value and the automatic turn-off of the display (phone sleep mode) was disabled. Our rendering and Wi-Fi consumption values closely reflect those published at [75], [3] and [30]. Our own measurements (see Table 4.2) show lower power consumption than estimated in these works, but have identical relative correspondence. This is likely due to the lower per-instruction power consumption budget of novel mobile devices.

For video streaming, bandwidth and power consumption do not depend on the number of rendered triangles, because we assume them to be processed at a sufficiently fast server.

	Female face	Male face
Triangles	8864	6352
Flat Shading	23.70	33.32
Smooth Shading	23.69	33.42
Flat, Directional Light	12.56	15.77
Smooth, Directional Light	12.58	15.76
Smooth, Point Light	3.76	5.77
Smooth, Directional, Textures	12.42	15.55
Flat, Directional, Textures	12.45	15.69

Table 4.1: The speed of face rendering in frames per second (FPS) depending on lighting, shading and texture settings

	Consumption
OpenGL rendering (8192 triangles), WiFi on	899 mW
Video streaming WiFi (100 kb/s)	1144 mW
Video streaming EDGE (100 kb/s), WiFi off	2252 mW
Playing pre-downloaded video, WiFi on	752 mW
Display on, WiFi on	402 mW
Client voice recognition (PocketSphinx)	433 mW
Server voice recognition using WiFi	1659 mW

Table 4.2: HTC Touch Pro power consumption

However, highly textured models can negatively affect the video-compression rate.

In cases where the 3D model is rendered on the client at stable FPS, power consumption rises with the number of triangles because every triangle needs some CPU instructions to be processed. Although we performed measurements with only three different sizes of models, results show that we can expect power consumption to grow linearly with the number of rendered triangles.

The measurements demonstrate that video-streaming power consumption is about twice that of rendering power consumption. A typical 1340 mAh / 3.7 V battery supplies 260 minutes of video streaming or 460 minutes of rendering of a high-detail (2000 triangles) scene.

Mobile device energy-efficiency computational tradeoffs are set to continuously improve, as reported in [55]. The number of computations per kWh doubles approximately every 1.6 years, which is a long-term industry trend. Therefore, the power needed to perform a task requiring a fixed number of computations will halve every 1.6 years, or the performance of mobile devices will continue to double every 1.6 years, while maintaining the same battery lifetime. Mobile wireless interfaces follow the same trend due to the vast processing required [89, 103] and are therefore unlikely to change the above balance favoring more computing on the mobile client rather than network data streaming.

4.4 Architecture Discussion and Selection

Various applications and mobile phones have different needs. Hardware performance of mobile devices differs greatly. That is why we decided to support both server and client speech recognition. We prefer server-side speech recognition over the client-side due to the limitation of memory and computational power of present mobile devices.

Solutions for speech recognition on mobile phones have lower quality than on servers,

which possess more resources and produce more natural speech dialog. Speech recognition is also memory- and CPU-intensive and both these resources are required for rendering. However, with future increases of computing power of mobile devices, we expect this to change in favor of client-side recognition.

Our video-streaming experiments have shown that latency of up to 400 ms may occur between user input and response from the server. According to this and the power consumption estimates and tests in Section 4.3, an architecture with graphics rendered on the mobile phone appears more convenient and efficient than one with the video streamed.

We prefer and support 3D-rendering and speech synthesis to be performed on the client only. It reduces client power consumption and connection-bandwidth needs, and is also more flexible in terms of user interaction and animation synchronization. Speech synthesis can be performed with sufficient quality on the more powerful mobile phones.

Therefore, we recommend creating applications with server speech recognition, application logic and client synthesis and graphics rendering (see Fig. 4.3).

4.5 Synchronization of Face Animation with Speech

The synchronization process is shown in Figure 4.6. Text is sent to the Text-to-Speech module where the synthesis is performed. During the speech-synthesis process, information about each generated phoneme and its duration is logged. While the audio wave data, created during the process, do not require any further processing and are directly saved into the audio stream, the logged phonemes and durations are passed to the conversion (Phoneme to Viseme Conversion). This conversion translates every phoneme to the relevant viseme (basic unit of speech in visual domain).

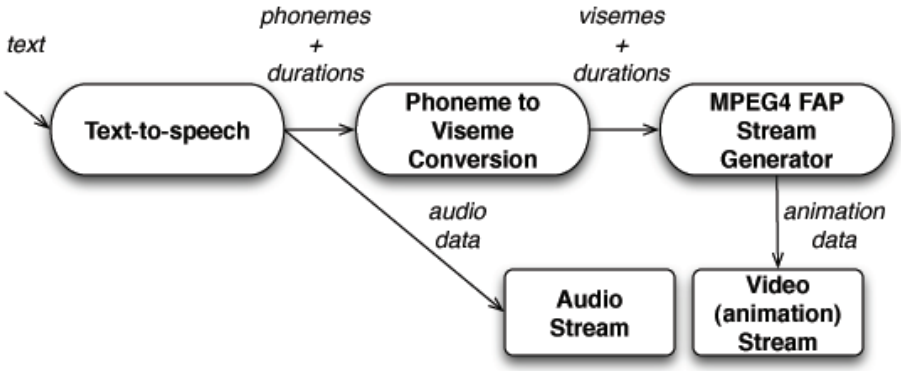

Figure 4.6: Process for generating face animation based on phoneme duration

Finally, based on the visemes and the timing information (durations), Facial Animation Parameters (FAPs) are generated and saved as animation streams. The synchronization of face and voice is then guaranteed when both streams are played simultaneously.

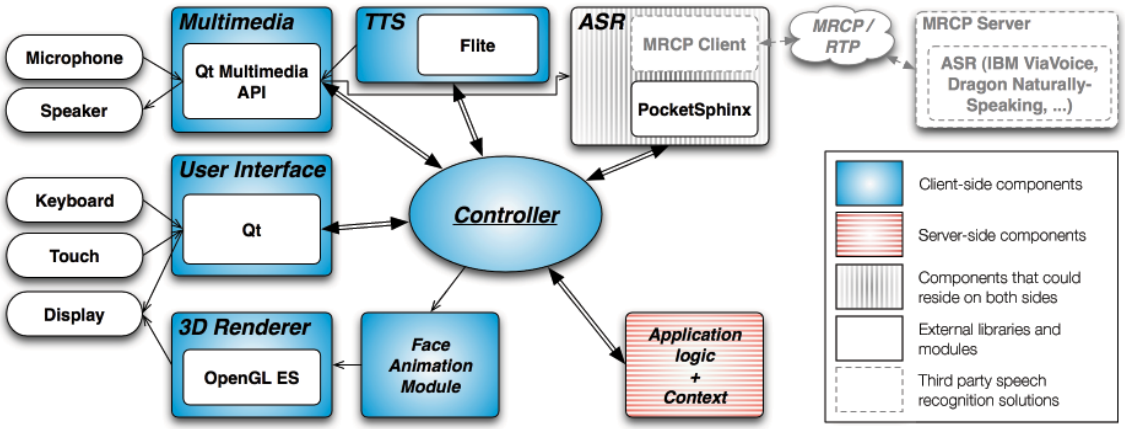

Figure 4.7: Final framework architecture

4.6 Framework Implementation

On the basis of the above findings we designed and implemented a platform-independent framework for creating talking-head applications for mobile devices. We chose the Qt library [96] for the user interface development and as a base for the entire framework for its flexibility and cross-platform portability. The framework is divided into several software modules and components (see Fig. 4.7).

The modules User Interface, 3D Renderer and Multimedia are responsible for interaction with the user in both the visual and acoustic domain. Rendering of 3D contents is performed by OpenGL ES [56] as discussed in section 4.2.3.

Face animation is generated and processed by the Face Animation module. We chose to use keyframe animation as defined in MPEG4 Facial Animation standard [84] (MPEG4 FA) for the animation of talking head and for a face-model features description.

For that purpose we modified and optimized the Xface [11] library to allow running on mobile devices and platforms. This library provides an API for the MPEG4-FA-based animation and the tools for face-model description. The Xface library also contains a parser of the SMIL-Agent [13] (Synchronized Multichannel Integration Language for Synthetic Agents) scripting language. It is an XML-based scripting language for creating and animating embodied conversational agents.

We use this language for creating dialogues between the user and the talking head. The application is then created by connecting SMIL-Agent scripts into a graph, where the nodes correspond to SMIL-Agent scripts and edges to user decisions (see Fig. 4.9).

Speech recognition and synthesis is provided by the Automated Speech Recognition (ASR) and Text-to-Speech (TTS) components. Both components have universal interfaces for supporting different engines via plugins. Our framework has built-in support for the Flite TTS engine [16] and the PocketSphinx ASR engine [43]. However, support for other engines is feasible with minor effort.

Moreover, the framework also contains Media Resource Control Protocol (MRCP) client for speech recognition, so any existing MRCP server with ASR media support can be used for speech recognition. While the ASR component may reside either on the client or server side, TTS must reside on the client side only, due to the necessity of synchronization of face animation and voice.

Application logic and context (e.g. user's session) is handled on the server side. The client communicates with the server using standard HTTP requests and responses. A standard web server is used for this purposes, but instead of HTML output the SMIL-Agent script is used as a response.

Applications created by our framework run on Windows Mobile, Symbian platforms, desktop Windows, Linux and Mac OS (separate source code compilation for each of the platforms is required). We are currently working on support for the Android, iPhone and MeeGo platforms.

Using our framework we created two example cross-platform applications. The first is a virtual customer–care center and the second is a virtual shop (see Fig. 4.9). The applications use talking heads generated by FaceGen and are capable of rendering an animated head model with 1466 triangles (see Fig.4.1 and Fig.4.8). The rendering speed of the applications is above 15 FPS (usual mobile video capturing framerate).

4.7 Chapter Conclusions

Using our framework we demonstrated that as mobile clients become more powerful, real-time rendering of a voice-interactive talking head is within their reach and we expect a boom in voice-interactive 3D mobile applications in fields such as entertainment, commerce, education or virtual assistance. The client-server architecture with local rendering and synchronizing 3D and audio components and remote logic control and speech processing allows applications to be less power-hungry with improved quality of virtual-character interaction.

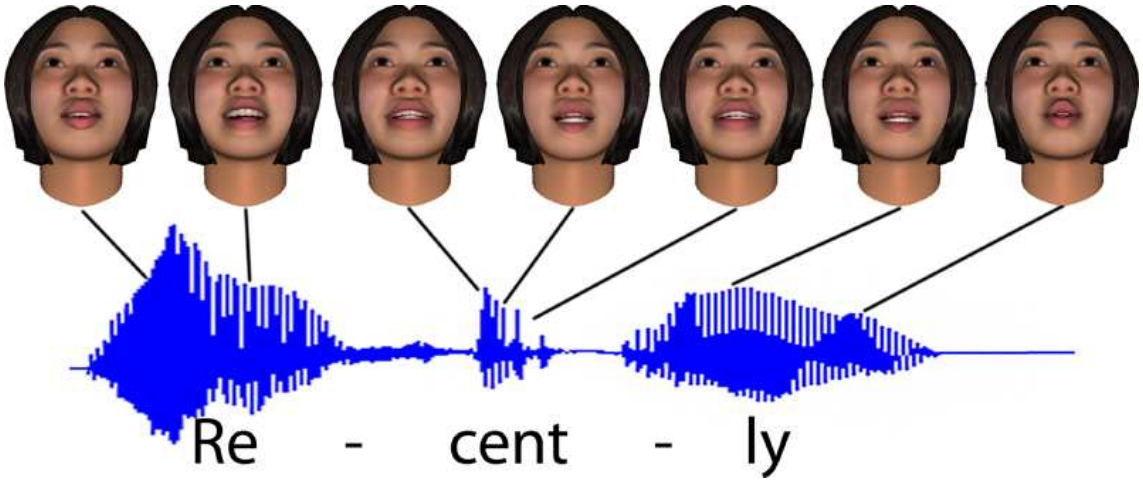

Figure 4.8: A synthetized word "Recently" contains three syllables (down) and it is visually represented by seven visemes (up). The viseme position in the timeline is set by the speech synthesizer. During the animation process the polygons of the model blend between adjacent visemes.


```
<par system-language="english">
  <speech channel="face" id="speech1">
    The tariff has been activated.
    Thank you for using the virtual operator.
  </speech>
  <seq channel="face" >
    <speech-animation affect="Rest"/>
    <speech-animation affect="SmileClosed"/>
  </seq>
</par>
```

Figure 4.9: An example of a created application – Virtual mobile phone operator. A snippet of our server application logic scripting – decision tree map (up) and corresponding script using XML based SMIL-Agent [13] scripting language (down, simplified)

Today, mobile-phone speech-application developers must deal with many platform-dependent interfaces. Speech application development can be facilitated by integrating synthesis and recognition libraries to the mobile operating systems. (Currently only Apple iPhone OS and Google Android OS support native speech synthesis.)

In our future work we plan usability testing of performance, voice recognition accuracy and user emotional response. We also hope to focus on the upcoming operation systems from Windows Phone [73] family that supports both speech synthesis and speech recognition through classes that are also a part of .NET Compact Framework 4.0 [122].

In the area of distributed architectures, we intend to enable the easy provision of mobile talking-head applications using cloud services. We see the future in such applications because they offer reduced server cost (paid incrementally as a utility), better reliability (automated server duplicating), flexibility in computation power and storage space, highly automated server maintenance, scalability, and allowing software developers to focus more on their core work.

The main challenge will likely be portability, as cloud application must be in a special form (e.g. .NET managed code for Microsoft Azure [72]) and we anticipate certain difficulties in porting current server applications to the cloud.

In our future work, we intend to investigate further reduction techniques as part of our ongoing effort to design an open platform for development of talking-head applications on mobile phones (using the Xface framework developed by Balci [10, 11]).

Part II

Collaborative Computer Graphics in Distributed Environments

Part II is based on our work published in [A.3], [A.4], [A.6] and [A.9].

Due to the exponential growth of mobile data traffic and bandwidth-hungry applications, there is a demand for new approaches to access content. We take an experimental approach to design and implement a cloud-controlled system for device-to-device (D2D) content distribution. Our objective is to reduce the traffic load of the cellular network by dynamically distributing the content from the cloud to a subset of subscribed users and allow these users to spread the content using D2D communication.

We investigate strategies for distributing the content with the cloud logic. We implement our system using real cloud service and mobile devices, evaluate our system using a video application running on smartphones and compare our solution with alternative approaches without the cloud infrastructure.

We take practical design aspects into consideration and study how different levels of feedback from the users impact performance. We show that solutions controlled by the cloud are more efficient in terms of traffic offload than approaches without cloud logic.

Chapter 5

Collaborative Device-to-Device Video Streaming

5.1 Introduction

A recent forecast study made by Cisco Inc. [25] shows that mobile phone workloads on cellular networks will double or triple every year. This dramatic traffic growth is driven by mobile video streaming, which is forecast to reach 69% of all traffic by 2018. Although the problem can be postponed by building the next generation of cellular networks, it is expected that the principal challenges will remain the unchanged [44].

As a limitation of Bluetooth technology, a device cannot accept a Bluetooth connection while (a) it is trying to connect to another device, or (b) while it is scanning for nearby devices. We then must consider that, when either (a) or (b) is concluded, the device waits for a random-length interval t for other incoming connections before actively scanning, where $t \in [t_1, t_2]$ and $(t_2 - t_1)$ is the listening period.

In our work, we focused on devices with two wireless network interfaces – one for connection to the Internet (e.g. cellular connection or Wi-Fi), and another for opportunistic communication to nearby devices on a device-to-device basis (D2D; e.g. Bluetooth or Wi-Fi Direct).

Recently, it was experimentally shown that such devices can cooperate to download video content and share it using their opportunistic wireless connection, thus addressing the cellular bandwidth crunch problem [51]. This approach is of interest for those applications where a local group of users, carrying their own mobile devices, expects to watch a popular Internet video (e.g. a live streaming of a popular sports event or an educational video that is watched by all students in a class).

While the above approach is more effective than using the current naive way of each device independently downloading video using its own cellular connection, several open questions in the architecture design and the implementation remain. First, it is still unknown how to conveniently deploy a system in the cellular network backhaul to deliver content. Second, users are nomadic and the available bandwidth of D2D communication can change greatly over time, thus affecting opportunities to offload traffic to D2D links. In order to answer to these challenges, we make the following contributions:

- We design a cloud service that coordinates D2D dissemination steps, based on inputs such as which device received which part of the content and information from opportunistic wireless network topology.

- We introduce several novel cloud–based strategies to reduce the load of the cellular networks and study how varied levels of inputs/feedbacks from mobile devices affect overall offloading capability.

- We implement our architecture, using Windows Azure cloud service and create an app running in smartphones and test our implementation in representative experiments.

We measure the performance of our methods with up to 16 smartphones and further emulate environments where congestion of opportunistic D2D channels may occur. In our emulation environment in a controlled setup, we achieve up to 71% of saved bandwidth with all the phones in D2D range. Our practical experiment with fewer phones in D2D range shows that users can save 39% on average and 53% in peak values.

5.2 Related Work

5.2.1 Opportunistic Content Sharing

The principles of cooperative techniques to disseminate content have been presented in [33], motivating cooperation will become one of the key technologies enabling improved cellular networks. Ramadan et al. [97] described an experimental cooperative video-streaming architecture with mobile devices sharing a single access point and D2D connectivity. Differently from [97], we designed strategies to disseminate the content under nomadic users and variable D2D network topology and congestion, testing these strategies with experiments.

The authors of [50, 65] studied a cooperative technique for opportunistic podcast and feed distribution. In their work, they assumed there is absence of cellular infrastructure, while we assume that this infrastructure exists, but the bandwidth does not suffice to disseminate popular content to all subscribed users.

The research work [42] allowed single video content to be received by multiple mobile devices using multicast from a common base station. Those devices outside the range of the base station can receive content only by using D2D opportunistic communication. Multicast increases the cost of complexity of the cellular base stations and requires traffic to be sent at lower rates [51].

Jung et al. [48] proposed a mobile system for collaborative bandwidth sharing that reduces the required cellular downloads. Based on decision tables, users decide whether or not to help other users download certain content and how much it should help.

In our design, we consider that all devices help to disseminate content and rather focus on strategies that reduce the cellular overload as much as possible. Our work shows that this comes at negligible cost in terms of fair download of content from the cellular network. All of the above works focused on groups of cooperating mobile devices without implementing a central coordinator in the cloud.

5.2.2 Comparison of Dissemination Techniques

Most of dissemination techniques that uses two wireless interfaces for communication were studied through simulations. The only related work that implements dissemination techniques is from Keller et al. [51], proposing a system called MicroCast for cooperative video streaming using cellular connection and Wi-Fi peer-to-peer connection. Their algorithm assigns the next segment of a video to be downloaded to whichever phone has the smallest set of segments to download from the cellular network.

In our system, we take a step ahead and consider practical problems such as the congestion of D2D communication, an adaptive algorithm that can choose the exact number m of devices such that the saved bandwidth is maximized.

Wirtz et al. [123] proposed a system for opportunistic mobile networking with a cloud component called ICON. Our approach differs from the previous two works by using techniques to avoid the congestion of opportunistic communication. A theoretical and experimental analysis of opportunistic dissemination algorithms was made by [105]. They proposed two variants of the opportunistic dissemination algorithm and compared them with the traditional client-server architecture.

Han et al. [41] made a case study for cellular traffic offloading using opportunistic

connections. They proposed heuristics to select the target set of users in order to minimize the cellular data traffic. However, they did not look at strategies during the dissemination period. Their analysis was evaluated by means of simulations, rather than studying and deploying a cloud-based infrastructure and experimental testbeds as in our work.

A peer-to-peer mobile system for video streaming with guaranteed quality was proposed by Navid et al. [1] and by Wihtbeck et al. [121]. They considered a group of co-located peer devices with dual-wireless interfaces that desire to receive synchronously a live content stream divided into chunks. Their algorithm decides which peer broadcasts a chunk on an opportunistic channel at each time and how long the transmissions should take place for each block. They then present an algorithm that ensures that quality-of-service targets can be met for each device.

5.2.3 Techniques for Avoiding Congestion

The authors of [106] proposed a strategy for message delivery in a delay-tolerant network, which alleviates congestion in intermittently connected mobile wireless networks. [100] described a set of algorithms for handling storage congestion by migrating stored data to neighbors. The algorithms decide which messages should be migrated to which neighbors and when. However, this approach works without cellular network infrastructure, without guaranteed delays and does not consider the problem of alleviating the cellular network load. Their experiments showed that the approach can improve the message completion rate by as much as 48% for some storage-constrained networks.

Burleigh et al. [18] investigated cases where network applications must communicate across disparate networking environments, often with high latency and potential for congestion. The authors presented a congestion avoidance and control mechanisms and promote carefully engineered congestion avoidance in networks.

5.3 System Architecture

We first introduce the scenario which is objective of investigation in this chapter and then present our system architecture.

5.3.1 Scenario

In this work, we suppose that there are N smartphones that subscribed to a common content. The phones may retrieve content from the cellular network while they are able to

perform opportunistic communication using their short-range wireless interface. In order to disseminate the content, we consider that such content is initially given to m devices that spread it throughout the opportunistic network to devices without such content.

When the deadline T_c for receiving the latest data content becomes close, there are m' devices that have not yet received the data chunk through either cellular networks or opportunistic communication. These devices request the chunk to the cloud via cellular infrastructure. This allows for *guaranteed delays* of the content delivery and to use an application like video-streaming without interruption in the service.

Overall, the total number of data transmissions through cellular network would be $D = m + m' \leq N$. We study the design of a system architecture that leverages the potential of the cloud to control opportunistic dissemination. The goal is to offload the traffic M from the cellular network as far as possible, while meeting the requirement that all devices have received the content before the specified deadline T_c.

In order to design such a system, we consider an architecture that is made of a cloud part and a client mobile part. A schema of the whole system is shown in Figure 5.1. In the following sections, we discuss the components of our architecture.

5.3.2 Media Recording and Subscription Service

The cloud part of the distribution system contains a media recording service that is capable of listening to online video streams on the Internet. Because mobile devices do not handle all video formats, the media recording service also converts the video to a specific compression format and splits its length into fixed intervals. We refer to these short video files as *chunks*. The media recording service stores them afterwards in a cloud virtual drive called Chunk storage.

A newcomer mobile device can register to desired channels using the Subscription service, see Figure 5.1. One device can be subscribed to multiple channels. The subscription preferences of the device are stored in the Subscription database.

5.3.3 Dissemination Service

The central component of the cloud is Dissemination service. Its main goal is to select the m devices that will be used for initial content spread. The initial spreading time of chunk k to m devices is denoted *spread period*, as shown in Figure 5.2. According to the specific strategy, the dissemination service might also provide additional information.

Once this injection from the cloud is concluded, the devices enter the *dissemination period*, which is the time T designated for D2D dissemination of the chunk to other devices.

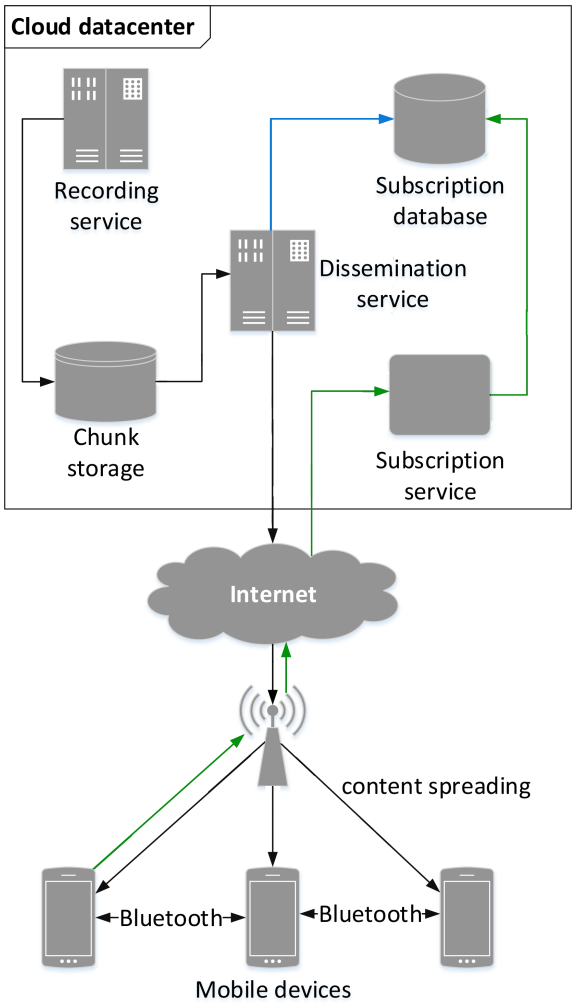

Figure 5.1: Schema of the cloud-based distribution system; Black – streaming data flow; Green – new content request from the device on the left; Blue - a database query

In our work, we use Bluetooth for D2D communication. Each mobile device attempts to connect to another device within its Bluetooth proximity. In case of the successful establishment of a connection, the two devices compare their lists of chunks that have previously been downloaded.

In order to increase the transfer rate, the dissemination periods of different chunks in a video-sequence can overlap, so it is possible to share more than just the last chunk. The overlap of different chunks is shown in Figure 5.2. A random order for delivery is defined for chunks present in only one of the two devices. The devices disconnect when there are no other chunks to be shared, or when their connection has been interrupted due to a communication failure. After that, both devices begin searching for another device to connect to.

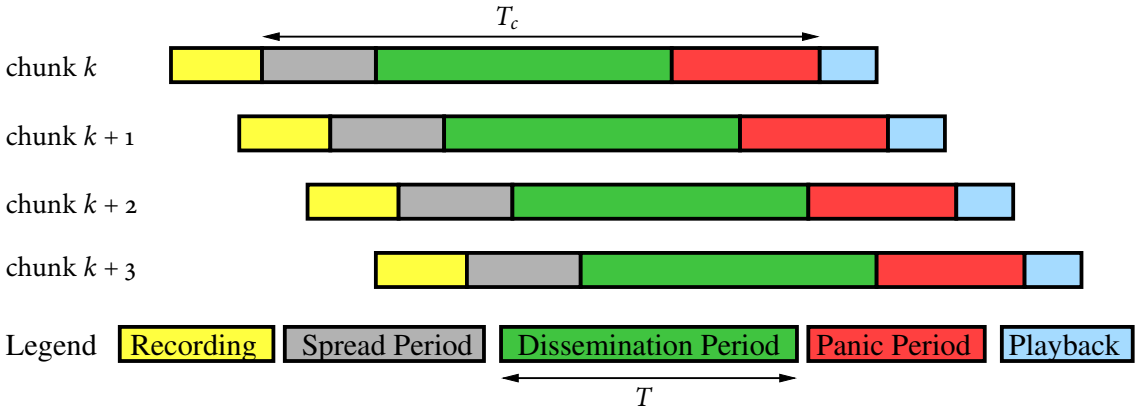

Figure 5.2: **Timeline graph for content distribution**: Each line represents the distribution of one content chunk. Note that the dissemination periods of chunks can overlap.

All deliveries from the cloud take place either before (m deliveries) or after (m' deliveries) the dissemination period. This has the advantage of avoiding monitoring how information spreads during the dissemination period. We use the term *dissemination deadline* to denote a point in time when the time for opportunistic dissemination ends for a particular chunk. After this moment, the devices must utilize their cellular connection to download the chunk. We refer to the time interval after the dissemination deadline as *panic period* (see Figure 5.2), because the devices use their costly connection to download the chunk shortly before its playback should begin.

A detailed example of different phases of content distribution for video chunks is illustrated in Figure 5.3, and is described as follows:

1. The cloud service begins recording a new video chunk from a web stream;
2. Device A begins receiving the recorded video chunk from the cloud from cellular connection or Wi-Fi;
3. We enter in the *opportunistic dissemination period*;
4. Device B receives the video chunks using an opportunistic wireless connection;
5. We reach the *dissemination deadline*;
6. We enter into the *panic period*, and Device B begins receiving the missing video chunk from the cloud from cellular connection or Wi-Fi;
7. Finally, the devices begin displaying the video;

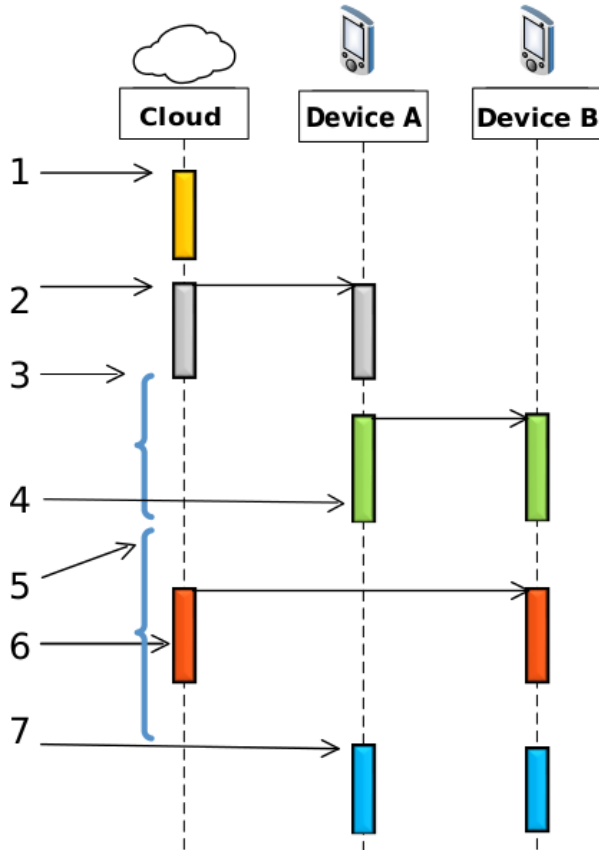

Figure 5.3: Example of communication for content distribution (UML interaction diagram)

5.3.4 Strategies

In this chapter, we investigate two general types of strategies that aim to reduce the traffic load of cellular networks:

- *Initial spreading strategies*: decide how many (m) and which devices should be selected for initial deliveries, when a new chunk is available (Section 5.4).
- *Dissemination strategies*: determine when and which devices should connect for D2D communication during the dissemination period, and which data they should share (Section 5.5).

We divide the strategies into those that utilize distribution logic in the Dissemination service component in the cloud and those that use distribution logic only in client mobile devices. One benefit of the cloud logic is that it can collect timely information about opportunistic network topology and use this information for the next deliveries through the cellular network.

In particular, we use the control connection from mobile devices to the cloud Dissemination service to report the list of other mobile devices that are visible in an opportunistic range. The distribution strategy subsequently uses this feedback to optimize the delivery of chunks.

The distribution strategies are described in detail in the following sections.

5.4 Initial Spreading Strategies

We introduce three strategies to initially inject copies of the chunks to m mobile devices during the Spread Period. In these strategies, we randomly select the m devices. This guarantees that there is no unfair usage of cellular network (e.g. some users using more their cellular connection than others for the same video content) as well as unfair internal battery depletion among the devices caused by higher use of cellular interface.

In the strategies where we have feedback concerning the D2D network topology, we further make sure that there is one device that receives the chunk per each connected component. This guarantees that all devices from one connected component of the corresponding D2D topology graph have the chance to receive one copy of the chunk before the end of the dissemination period. It avoids the problem that all the devices of one connected component may all need to download the chunk from the cloud during the panic period.

Next, we describe how to select the number m of devices.

5.4.1 Fixed Ratio Spread

As a simple distribution strategy for initial spread, we distribute the content to a fixed ratio of the number of devices subscribed to the content. (In other words the number of initially spread packets m is constant.) The benefit of this strategy is that it is easy to implement and does not need feedback from past distribution attempts (i.e. past chunks delivered).

With some minor modifications, this strategy can also be implemented in scenarios that do not use distribution logic in a cloud component (and thus client-only). In this case, at the beginning of the distribution of each chunk, a mobile device downloads the chunk in the initial spread period with given probability. In terms of performance, the difference between implementing this strategy with a cloud logic and with client-only approach is that the latter does not know the D2D network topology, and thus it may be possible that none of the devices in the same connected component decide to download the content through the cellular network.

Fixed ratio spread strategy has the disadvantage that a constant m may be sub-optimal

to save cellular bandwidth. For instance, in the case of too low initial spread, a high rate of chunks m' may be downloaded in the panic period after the dissemination deadline, which may result in total number of deliveries from the cellular network D which is higher than with some adaptive strategy.

5.4.2 K-armed Bandit Strategy

The second strategy builds on a classical problem of exploration vs. exploitation trade-off [112]. It attempts both to maximize the short-term reward based on current findings and to discover optimal parameters for long-term rewards in future. In the original K-armed bandit problem a gambler faces a row of slot machines and has to decide which machine to play in each of a number of games. Each machine provides a random reward from a distribution specific to that machine. The gambler's objective is to maximize the sum of the rewards earned through a sequence of lever pulls.

We identified an analogy between the K-armed bandit problem and our chunk distribution problem. The number of games in the K-armed bandit problem corresponds to the number of data chunks in the video stream, as they both represent the number of rounds in the problem. The number of levers K corresponds to the length of the chunk sequence in the video. The chosen lever can be interpreted as the number of initially disseminated data chunks. The gambler's reward is analogous to the number of cellular chunk deliveries, which has been saved by using opportunistic data saving.

5.4.3 Initial / Deadline Balance

The dynamics of the content distribution system is subject to changes. For instance, inter–contact time statistics may vary depending on the time of the day [20]. Therefore, the initial spreading strategies should adapt its decision on the value of m according to latest behaviors of the system. We then take advantage of the fact that the cloud dissemination service can record the number of chunks m' that were distributed after the former dissemination period. Thus, in order to deliver the new chunk k, this strategy includes a feedback mechanism that accordingly adjusts the number of initially spread chunks.

In order to select m, this strategy considers that a large number of downloads after the dissemination deadline might indicate insufficient initial spread of chunks by the cloud, and vice versa. According to the strategy, we then have to inject the chunks from the cloud in such way that we balance the number of initially spread chunks with the number of chunks distributed after the dissemination deadline (that is $m = m'$).

With Initial / deadline balance strategy, we avoid the situation in which only a few users receive the content (by delivering a number of chunks m through cellular communication at the beginning), as well as the situation in which few users miss out (by delivering through cellular communication m' chunks at the end of the period). Our experiment results approve this theory and show that feedback balancing of initially and deadline distributed chunks yield a higher number of peer-to-peer disseminated chunks compared with a fixed number of initially spread chunks.

5.5 Dissemination Strategies

We present three strategies to disseminate the content chunks during the dissemination period.

5.5.1 Client-only Dissemination

After the initial spread phase, each mobile device attempts to connect to a random device within its Bluetooth proximity. In the event that connection is successfully established, the two connected devices compare their lists of downloaded chunks stored with ongoing dissemination period. The client-only dissemination strategy uses the cloud only for downloading the chunks, both in the spread and panic periods.

5.5.2 Cloud-based Dissemination

In this strategy, each mobile device reports every chunk downloaded from D2D dissemination to the cloud service. When a mobile device completes its scanning of nearby Bluetooth devices, it downloads a list of their chunks from the Dissemination service in the cloud. This ensures that the mobile device will avoid connecting to other mobile devices that have no chunks available for sharing. On the other hand, the increased communication with the Dissemination service means some additional signaling load.

5.5.3 Adaptive Cloud-based Dissemination

In those cases where many Bluetooth devices are in proximity, their communication may interfere and, as a result, there will be more unsuccessful connection attempts. To deal with this situation, we propose a protocol that reduces the frequency of communication attempts in cases when there are too many connection failures, which increases the random period when the devices are listening for incoming connections.

In the event of an unsuccessful Bluetooth connection attempt, this strategy will increase the listening period $(t_2 - t_1)$ by the multiplication-factor parameter α. In the event of a connection success, the strategy will decrease the listening period by the multiplication-factor parameter β.

5.6 System Implementation

In this section, we introduce the implementation details of our cloud-controlled system for D2D content distribution.

5.6.1 Cloud Services

All our cloud services are based on Microsoft technologies and are hosted in the Windows Azure environment. Applications for Microsoft Azure are written in C# using .NET libraries. The Recording service and the Dissemination service run as worker–role instances. Chunk storage is implemented like cloud blob storage.

We use Azure Mobile Services technology for the Subscription service, which allows seamless integration of cloud services into mobile client code with support from development tools. The subscription database is based on Azure SQL Database (the cloud version of the Microsoft SQL Server).

A new device can register to desired video channels and the worker role of the Dissemination service is then notified about the new device and its content requests. It will then consider the new device in its future data delivery schedules.

When a new chunk is available, the device may be notified by the Azure Mobile Service that it may download the chunk from the cloud. Otherwise the device will download the chunk using opportunistic communication during the opportunistic dissemination period. If the content is not available after the dissemination period, the device will request it from the cloud.

5.6.2 Mobile Devices

We implement a video streaming app that runs on Windows Phone 8 mobile devices. We chose Windows for its better integration with Windows Azure cloud service. We used HTC8S smartphones to test the application.

For purposes of our research, we use Bluetooth 3.1 communication, which depletes the battery less than Wi-Fi Direct. (In addition, Wi-Fi Direct is not yet supported by the

Windows Phone 8 platform.). The video in the app is transparent to the actual wireless interface used to receive the content chunks. The app handles initial D2D discovery and can send signaling messages to the cloud, if requested by the strategy. Time synchronization is performed with Windows Phone time server.

5.6.3 Signaling

For the strategies with cloud logic, the signaling mechanism is controlled by the mobile device. When a new chunk is available, the device queries the Azure Mobile Service. The reply will state whether the chunk should be downloaded immediately (or not) from the cloud during the spread period. In the latter case, the device will try to get the content during the dissemination period. In case the device has not yet received the chunk by the end of the dissemination period, it will make a new query to the Azure Mobile Service to request the chunk. Therefore, at least one query and up to two queries per chunk are necessary.

Figure 5.4 shows the format of the request from the device to the Azure Mobile Service and the corresponding reply. Overall, the size of the signaling is such that it is largely negligible for reasonable chunk sizes, such as the ones for video content. The request from the device to the cloud (D2C) has a Strategy field, which indicates the strategy used.

Not shown in the figure, the last bit of the Strategy field is a flag, indicating whether the device is in spread period or panic period. Next, there is the chunk ID that the device is querying for download (Request Chunk). Depending on the Strategy field value, it will further send the list of chunks downloaded by the devices and available for

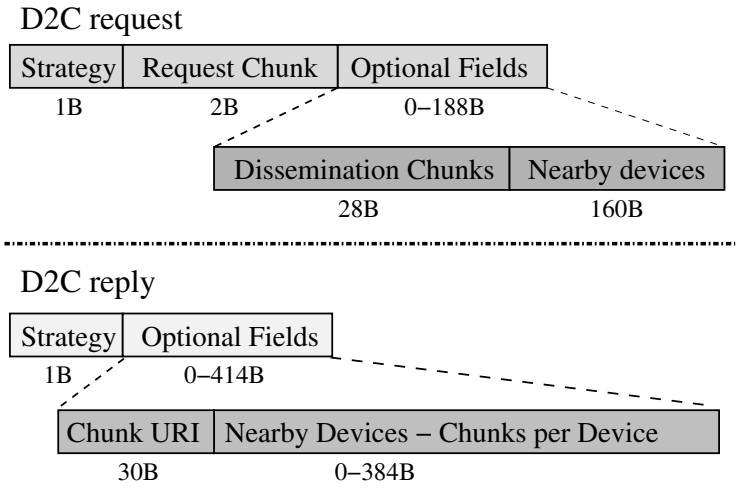

Figure 5.4: Packets format of signaling between the device and the cloud (device to cloud - D2C).

D2D communication (the ones that are currently within their dissemination period, i.e. Dissemination Chunks) as well as the devices in D2D range (Nearby Devices).

The first field of the response from the Azure mobile service is the Strategy field. As in the request, the last bit of the Strategy field is a flag. For a response, the flag indicates whether the device should download the chunk or not from the cloud. In case of a positive reply, the uniform resource identifier (URI) is given to the chunk to identify the path in Chunk Storage (Chunk URI). If requested by the strategy and if the device did not indicate that it is in panic period, a list of nearby devices and their chunks can also be provided (Nearby Devices - Chunks per Device).

5.6.4 D2D Communication

D2D packets are encapsulated by Windows Phone OS using a high-level API that abstracts the communication to a data stream. The communication is initiated by one device (device A) after an active scanning. Device A connects to a device (device B) which is in Bluetooth listening mode. The communication is composed by a handshake to exchange the set of chunks in dissemination period that are available and that are missing. By comparing the two lists, device A and B will then exchange the chunks missing to the peering device, if locally available. The chunks are sent according to a random order.

There are also some practical aspects to take into account in the implementation. In particular, the visibility relationship between mobile devices is not always perfectly symmetrical. This is because the devices scan for nearby devices at different points in time and other factors. Thus, in our implementation it suffices that one device reports it is in Bluetooth range to another and to include them in the same connected component of the D2D topology graph.

5.6.5 System Setting

The main parameters are summarized in Table 5.1. In our implementation, we use a content's deadline of $T_c = 100$ sec and a playback delay for the dissemination of 110 seconds. For higher content dissemination it is better to use longer playback delays because it takes several seconds to scan for devices withinin Bluetooth proximity or to create a Bluetooth connection. Content's deadline of 100 seconds allow devices to perform multiple attempts for content dissemination.

For the D2D opportunistic communication, we set the dissemination opportunity interval to $T = 55$ sec to provide enough time for D2D content dissemination. Because

Parameters used in experiments common to all strategies	
Video compression	MPEG
Video resolution	426×240 pixels
Content's deadline	$T_c = 100$ sec
Playback delay	110 sec
Recording time for each chunk	10 sec
Initial spread phase length	15 sec
Dissemination period length	$T = 55$ sec
Panic period length	30 sec
Playback time for each chunk	10 sec
Average chunk size	512 kB
Default listening period	$t_1 = 5$ sec; $t_2 = 15$ sec

Parameters specific to individual strategies	
Fixed spread ratio	25%
Exploration probability ϵ	10%
Adaptive factor α	1.15
Adaptive factor β	0.95
$\mathbf{K_p}$ (Initial/deadline balance strategy)	0.2
$\mathbf{K_i}$ (Initial/deadline balance strategy)	0.1176
γ (K-bandit strategy)	0.9

Table 5.1: Setting used in the experimental evaluation

this period is 55 sec and the recording interval is 10 sec, up to 6 chunks can be exchanged between two devices during one D2D connection.

Regarding the parameters K_p and K_i of the initial/deadline balance strategy, they must be chosen as a trade-off between a stable and reactive system, using Ziegler-Nichols rules [35] and imposing the stability constraint on the closed-loop gain.

Finally, for the purposes of testing and performance measurements, we use prerecorded video streams rather than live streaming. This approach circumvents any problems with the streaming source and the reproducibility of the tests. To this end, we use prearranged chunk files in Chunk storage.

5.7 Evaluation

We perform two different evaluations, first using a controlled setup, and then measuring performance with nomadic users.

5.7.1 Automated Testing System

In order to compare and evaluate the dissemination strategies described above, we implement an automated testing module that tests the performance of the Dissemination service. The module repeatedly sends a short video stream (10 minutes) to mobile devices. Logs from the devices are then collected at the end of each experiment. After each test, the module changes the dissemination strategy that is used, or its parameters. The module can also place a mobile phone temporarily outside the dissemination process for one test by sending a command to turn off its Bluetooth connection. This step enables measurements to be made for groups between 2 and 16 phones.

We run the experiments overnight to minimize interference from other 2.4 GHz radio sources. By subsequently analyzing the logs, we can compare the effectiveness of dissemination strategies under different conditions. The results of automated testing are shown in Figure 5.5 that displays the saved bandwidth over the number of devices used in the experiments. The saved bandwidth is defined as the amount of data in bytes downloaded over D2D communication over the total video size in bytes downloaded to watch the video.

We first analyze the result of initial spreading strategy. As a baseline strategy for the comparison, we use *Cloud-based dissemination* as our reference dissemination strategy.

Results of initial spread strategy in Figure 5.5 show that K-armed bandit strategy

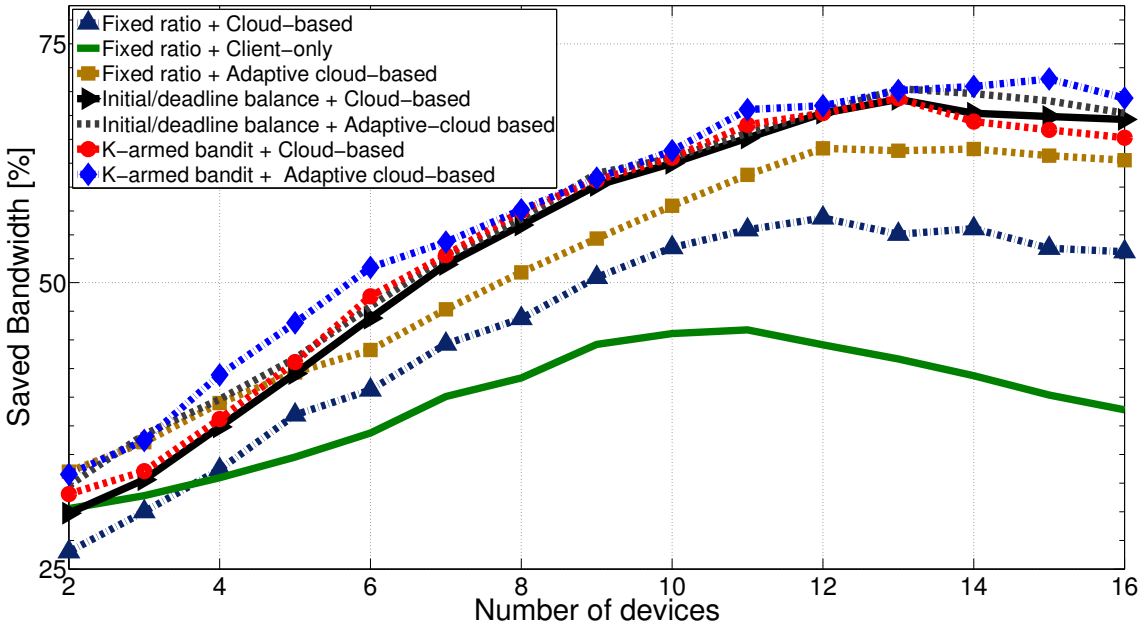

Figure 5.5: **Comparison of saved bandwidth for different strategies**: Each value is the average over three tests (each test is 10 minutes long). As you can see, for a larger number of devices the saved bandwidth is declining, probably due to wireless congestion.

and the Initial/deadline balance strategy achieve similar performance in terms of saved bandwidth. However, while the K-armed bandit achieves slightly better performance for a small number of devices in range, the opposite holds for higher numbers of devices. The reason the Initial/deadline balance strategy is slightly worse than expected is likely due to the fact that this strategy implicitly assumes that, at any point in time, the probability that there are two contacts (two D2D communications in parallel) is negligible. In reality, this hypothesis does not hold, due to protocol artifacts such as the fact that Bluetooth applies frequency hopping and thus parallel communications may occur on different radio frequencies. In contrast, K-armed bandit strategy does not rely on this assumption.

We further notice that a Fixed ratio strategy achieves poor performance and even that performance decreases for more than 12 devices. This is because the bandwidth available for D2D communication becomes a bottleneck and thus more injections in absolute in the spread periods have the effect of increasing the failed attempt of D2D communication.

We then study the dissemination strategy and show the results in Figure 5.5. As baseline for the comparison, we use *Fixed ratio spread* as our initial spreading strategy and look at which dissemination strategy helps reduce congestion of the D2D communication. We observe that Cloud-based strategies outperform Client-only strategies.

Not shown in the figure, results with Client-only get even worse with higher fixed spread ratios. For instance, we report 31.43% of saved bandwidth with 16 devices rather than 36.66% using a fixed spread ratio of 50% rather than 25%. The best performing dissemination strategy is the Adaptive cloud-based strategy.

Finally, we compare the initial-deadline balance and K-armed bandit, using the Adaptive cloud-based as our dissemination strategy. The plot in Figure 5.5 shows that the combination of K-armed bandit and Adaptive cloud-based strategies generally outperforms any other strategies, with up to 71% of saved bandwidth.

We analyze in more detail the performance loss for high numbers of devices. We define the congestion signal as the number (in percentage) of unsuccessful opportunistic connections between mobile devices over the total number of attempts.

Figure 5.6 shows that a Client-only strategy is greatly affected by a very high rate of congestion signals for increasing numbers of devices. Thus, the devices are not able to download the chunk from D2D communication, despite devices in range with a copy of it. In contrast, the adaptive cloud strategy shows a quite stable level of congestion signal, which explains why it outperforms other strategies in terms of saved bandwidth.

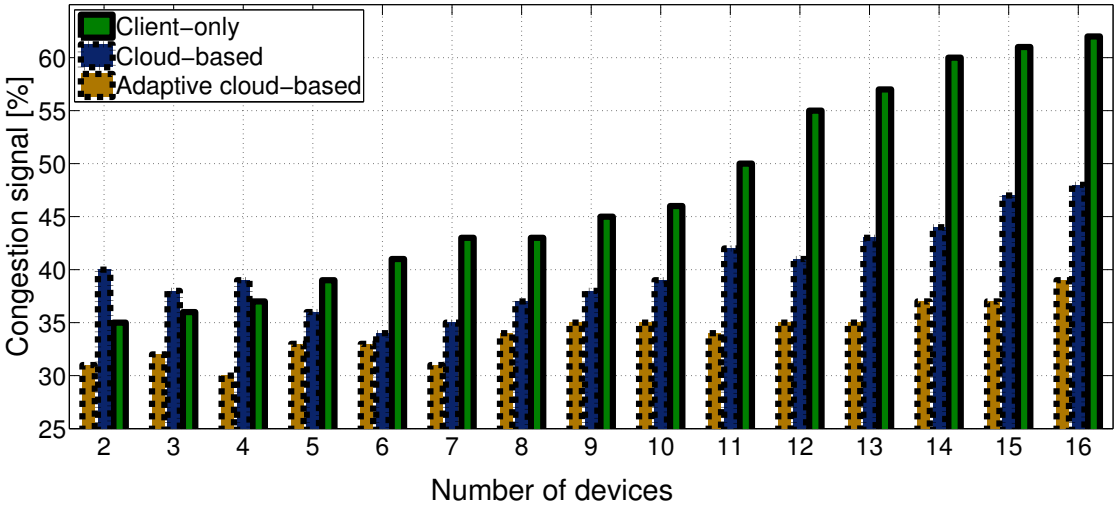

Figure 5.6: Congestion signal of D2D communication for the three different dissemination strategies

5.7.2 Evaluation with Nomadic Users

To evaluate our system in practical applications, we organized two experiments on different days with volunteers that carry a mobile device with our application. The volunteers have been working in the same building and carried their mobile devices all the time. In Experiment I, we had 16 volunteers and the test lasted for approximately 4 hours. In Experiment II, we had 12 volunteers over an 8-hour period. For Experiment I, we use a Fixed ratio as initial strategy and Client-only as dissemination strategy. For Experiment II, we use the K-armed bandit as our initial spreading strategy and the Adaptive cloud-based dissemination as our dissemination strategy.

Both experiments were performed during working time when most of the volunteers are inside the building or in its proximity. The devices streamed a video with a bit rate close to 400 kb/s (a usual bandwidth for a low-resolution YouTube video). The experiment attempted to emulate situations when multiple users want to access the same video content and may have the opportunity to be in D2D communication range. The logs of the experiments were retrieved manually from the devices after the test is over.

We first compared the Empirical Cumulative Distribution Function (ECDF) of both experiments in Fig 5.7. On the left of the figure, we show the saved bandwidth with nomadic users during the same period of time (from approximately 12:00 to 16:00). Each sample is the average saved bandwidth over a short time interval of ≈ 15 minutes. The figure clearly shows that Experiment II, using a combination of initial spread and dissemination strategies gave superior performance in the automated testing system in

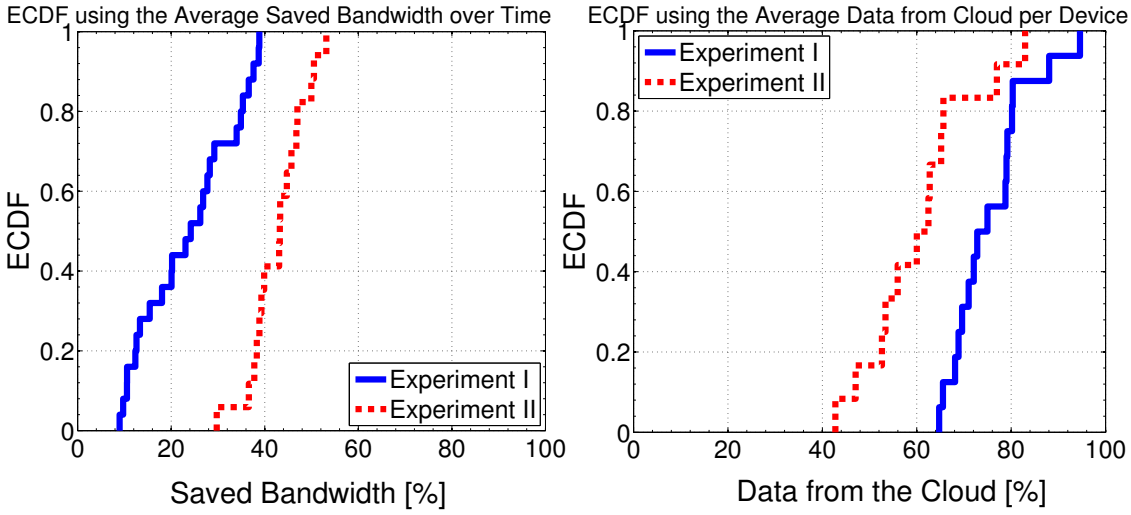

Figure 5.7: **Comparison of Experiment I and Experiment II** – On the left: ECDF of average saved bandwidth over 4 hours. On the right: ECDF using the average data downloaded from the cloud per each device.

the previous section and obtains significantly higher saved bandwidths than Experiment I. This results has been achieved despite more opportunities to share the chunks with D2D communication in Experiment I, which reports 3.53 devices in range on average, while Experiment II reports 2.52 devices.

We then studied the amount of data (in bytes) that each device downloads from the cloud over the total amount of data downloaded from both the cloud and D2D communication and compute how fair the injection of chunks from the cloud is using different strategies. For each device, we computed the average over the entire test, and plotted the ECDF of data downloaded from the cloud on the right of Figure 5.7. We measured a Jain's fairness index of 0.989 for Experiment I and 0.967 for Experiment II. This result shows there is a fair access to the costly wireless network interface (such as 3/4G). Concluding, a Cloud-based approach is preferable in terms of saved bandwidth and comes at negligible cost in terms of fairness.

In Figure 5.8, we then depicted the saved bandwidth, and the average number of nearby devices after Bluetooth scanning for Experiment II as a function of the time. The resulting saved bandwidth was 39% on average and 53% in two peak values at 10:45 and 13:45. The figure shows that the amount of saved bandwidth tends to increase and decrease according to the number of nearby devices in range, with a high Pearson correlation coefficient between the two variables of 0.79. As a result of this high correlation, we also observed that the strategy can adapt to variable conditions of the D2D network topology.

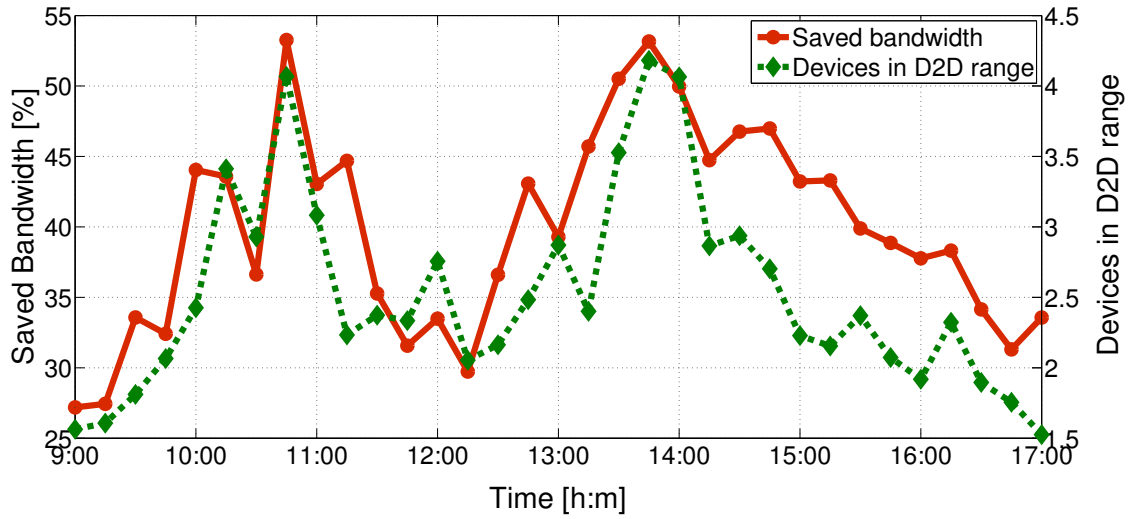

Figure 5.8: Bandwidth savings in the practical experiment for Experiment II and average number of nearby devices after Bluetooth scanning

5.8 Open Issues and Discussion

5.8.1 Distribution of Application Updates and OS Integration

Peer-to-peer wireless sharing is currently not integrated into phone operating systems. Windows Phone and iOS apps run in a sandbox that restricts some operations required for communication with other devices, especially when the user is not using the application. Application updates are also managed on the system level and cannot be managed by 3rd party code, however the approach can be implemented as a part of phone OS. We are aware that Windows Phone 8 OS uses a cloud pre-compiled code for particular device hardware and therefore devices with different hardware cannot share application updates. However this step is only optional and phones are capable of compiling the whole app from a hardware independent binary code called Common Intermediate Language. Windows Phone 7 OS use on-device code compilation from Common Intermediate Language.

5.8.2 Fairness

When multiple users use pay-as-you-go resources, we must consider fairness of cost distribution among the users. In some scenarios certain malicious users may misuse the paid resources of other participants. Our current implementation appears to be safe against such attacks on principle because users access the paid cellular connection only to download content they need. Therefore the users accessing the system will not pay more

compared to those not using it. Attackers may still misuse the system by downloading content through Bluetooth and saving some battery, e.g. by not sharing their chunks with others, but this would do no serious harm. However non-experimental applications should tackle this issue.

5.9 Chapter Conclusion

In this chapter, we have shown that cloud logic can help alleviate the saturation of cellular network traffic. Using the cloud, we implemented and experimentally evaluated a novel architecture to disseminate popular video content to subscribed users. Our measurements showed that opportunistic dissemination techniques utilizing a coordinating cloud service can achieve higher offloading rates than techniques compared to those without cloud logic. Use of cloud services also allows achieving better effectiveness in environments with large numbers of mobile devices where it avoids the congestion of wireless channels.

We envision that the inherent scalability properties of the cloud allows content providers to deploy multiple instances of media recording services for a large number of requested media streams and dynamically adapt the allocated network resources according to the number of subscribed users and the dynamics of D2D communication.

For common usage of delay-tolerant networks and opportunistic data offloading for mobile devices in developed countries, it is necessary to integrate support for this service to the phone operating system. If not, the technique will suffer from lack of supporting devices. A standardized dissemination Bluetooth protocol would allow cross-platform dissemination (e.g. Android – iPhone – Windows Phone) and thus significantly increase the group of involved devices.

In addition to media streaming, it would also be useful for RSS readers or application updates, because some applications are installed on many phones.

Part III

Virtual Cities on Mobile Devices

The content of Part III is based on our work published in [A.1], [A.3], [A.7], [A.8] and [A.9].

In this part we present novel techniques for implementing possibly infinite on-demand generated 3D virtual worlds in distributed environments. Our approach is useful under two scenarios:

1. A multiuser virtual world with mobile clients having sufficient CPU and GPU power but limited network speed. This reflects current mobile phones, tablets and laptops in areas without high-speed mobile connections or Wi-Fi connectivity.

2. Virtual world on-demand generation in a cloud environment that would be useful for scalable and massive multiplayer games.

If multiple independent generators create areas that overlap, our method ensures that the intersection of these areas will contain the same geometry for all of them. For this reason, we call our method *Stateless Generation*.

Chapter 6

Procedural Generation of Cities

In this chapter we describe approaches in procedural city modeling, workflow for procedural city modeling, procedural generating building and the previous state in real-time procedural city generation of cities.

6.1 City Modeling Approaches

There are two main approaches to procedural city modeling – behavioral and geometric. However recently, researchers succeeded in combining both approaches.

6.1.1 Behavioral City Modeling

The behavioral approach focuses on simulation of city development in time. The simulation begins with a city layout acquired from a map of a real city or form an artificial one. It is important to know the types of buildings, because they have a significant effect on city development. The systems usually distinguish at least three building types – residential, commercial and industrial. The city simulation is often restricted to a regular rectangular grid, so only a few behavioral modeling systems can work with arbitrarily oriented buildings. Visualization of the city is not usually too important. Some systems for behavioral city modeling use only simple 2D visualization. An external program (such as SimCity) can be used for the visualization. A terrain height/heightmap is usually not considered during the modeling.

Behavioral city modeling is usually used for city development prediction, urban planning or computer game simulations.

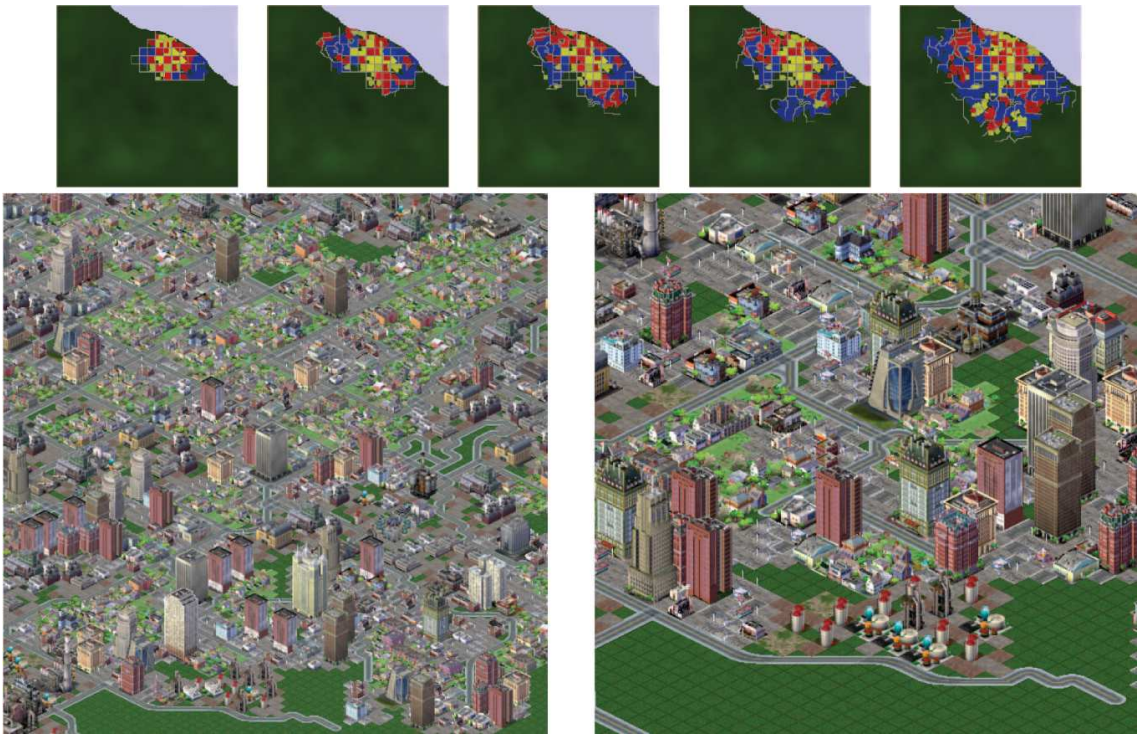

Figure 6.1: An example of behavioral city modeling published by Lechner et al. [63] (top) and its visualization using SimCity 3000 (down). SimCity 3000 is a commercial computer game developed by Maxis in 1999. Different colors represent different building types/land utilization.

6.1.2 Geometric City Modeling

The geometric approach focuses on the creation of a visually pleasant 3D city model that does not evolve over time. This approach is usually used for computer games that require a static city model. A realistic look of the city is the primary goal in geometrical city modeling. The modeling systems use a heightmap and may have arbitrarily oriented buildings. The output of the modeling software is usually a detailed polygonal city model with textures.

6.1.3 Combined City Modeling

In recent years some researchers have attempted to combine both previous approaches. In 2009, Vanegas et al. [113] presented a system that combines both the geometrical and behavioral approach in city modeling. Their approach combined both city evolution over time and a detailed realistic 3D look of the city. Weber et al. [120] presented a system in 2009 that can quite precisely predict city development from a city map or create an entirely new city.

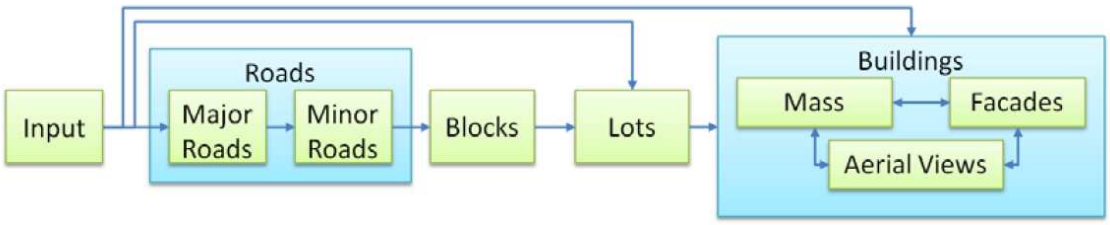

Figure 6.2: General pipeline for city modeling. [114]

6.2 City Modeling Workflow

This section presents the most common workflow for city generating, based on a street network. This approach was presented for the first time by Parish et al. [86] in 2001 and is now used in nearly all research works concerning geometrical procedural city modeling.

The key element in the workflow is a city road network that is modeled first. The road model is created using an L-systems [93] technique extended by context sensitive rules [86]. The L-systems were originally developed for procedural modeling of plants, but they can be used for road network modeling as well.

In 2006 Müller proposed a new language called CGA (Computer Generated Architecture) [76] that can be used for writing rules for procedural building modeling. The CGA language is also based on the L-systems, but has many substantial extensions. Since then the CGA language became popular in the procedural building modeling area, because the previous modeling techniques were not much suitable.

CGA rules can be retrieved from existing building using automatic, semi-automatic or manual approaches. For our experiments we prepared several CGA rules from photographs of existing buildings - see Figure 6.3.

We briefly explain some of the terms used in the area of procedural city modeling:

- **Major roads** usually represent highways or other large roads. In some modeling systems these roads must be manually placed. Creating the major roads is the first step in the city modeling workflow.

- **A quarter** is a land area enclosed by major roads (with no major roads inside). Quarters are usually subdivided by minor roads and populated by buildings.

- **A minor road** is a road inside a quarter used to subdivide the quarter. Minor roads are usually generated by context-sensitive L-systems.

Figure 6.3: **CGA rules created for existing buildings**; Left column: Original photograph; Right column: A building generated using the extracted CGA rules; We would like to thank Jakub Vampola for his work on preparing CGA rules of existing buildings.

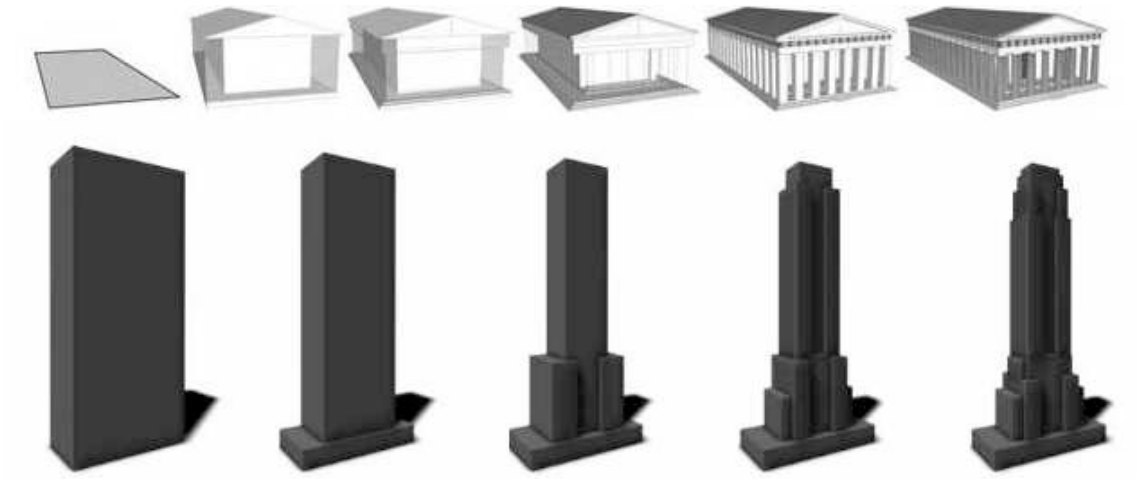

Figure 6.4: **Several generations of the buildings derivation**: Sometimes the interim models can be used for LOD. [32, 86] This approach cannot be used when the models are textured.

- **A block** is a polygonal-shaped land area enclosed by roads (with no roads inside). Usually several buildings will be generated inside one block lot. Block lots are usually subdivided to several building lots.
- **A building lot** is a land area dedicated for generation of a single building.
- **Building mass modeling** creates a rough geometric shape of the building. A building mass is usually generated by CGA grammar rules.
- **Building facade modeling** splits buildings to levels and creates wall geometry. Building facades are usually generated by CGA grammar rules.

6.3 Previous Work in City Modeling

The most advanced approach for procedural building generation was published by Müller et al. [76] in 2006, improving upon the previous method by Wonka et al. [124] in 2003. The lot and street geometry can also be generated procedurally. The first such algorithm for finite cities was published by Parish and Müller [86].

In 2003, Geuter et al. [38, 39] presented an algorithm for the on-demand generation of infinite cities in a regular rectangular grid. In their approach, the street network has to be aligned with the main axis, and all building lots must have the same square shape and size (see Figure 6.5). The visible buildings are determined and procedurally generated according to the viewing frustum. Each building lot is assigned an integer number according to its

coordinates using a hash function. This number is used as another seed for the pseudo-random building generation of that building lot. Some of these ideas are applied and extended in our approach. According to the viewing frustum, the visible buildings are determined and procedurally generated. Each building lot gets an integer number according to its coordinates using a hash function (see fig. 6.6). This number is used as a seed for the pseudo-random building generation of that building lot (see fig. 6.7). The generated buildings are saved into cache to save system resources.

Figure 6.5: Previous approach in infinite-city rendering published by Greuter et al. [38, 39] showing street level view. Note the regular rectangular shape of the street network.

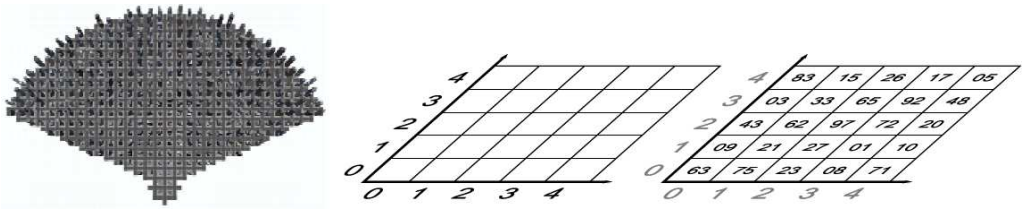

Figure 6.6: **Determining building generator seeds from coordinates**: The rendered buildings are determined using intersection of the viewing frustum and building bounding boxes.(left) A hash function assigns a seed for the pseudo-random building generator to each building lot according to its coordinates.(right) Source: [38]

A method for real-time generation of detailed procedural cities from GIS data was published by Cullen and O'Sullivan [27]. Their system uses a client-server approach, allowing multiple clients to generate any part of the city without requiring the full data-set. It creates the building geometry on-demand from the provided lot database and, in contrast to our work, does not address street and lot generation. Vanegas et al. [115] presented an interactive method for procedural generation of city parcels. They generate spatial configurations of parcels similar to real-world cities and support consistent lot locations

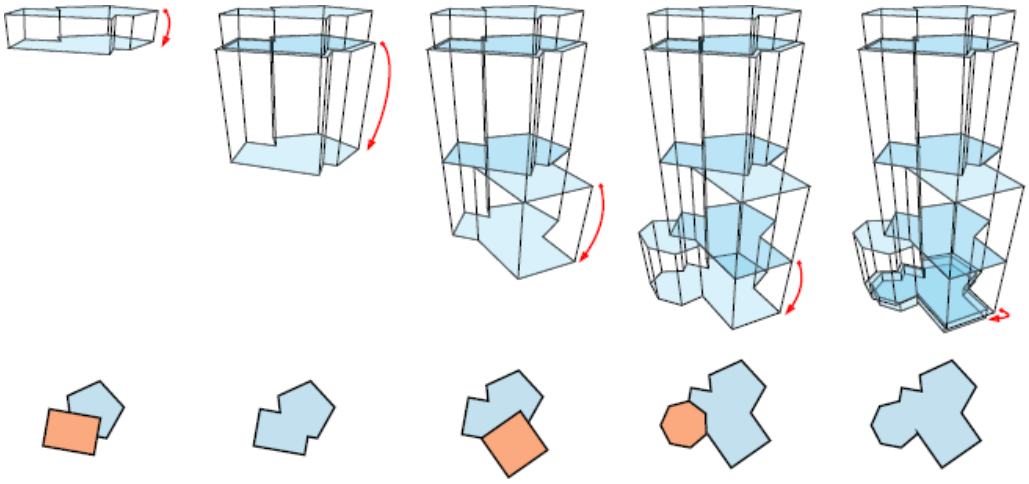

Figure 6.7: **Generating floor shapes**: In the method of procedural generation of buildings proposed at [38] the building is generated from top to bottom. The roof of the floor consists of two random primitive shapes. The top floor has the shape of a slightly enlarged roof. Each additional floor adds one random primitive shape to its shape. At the end of the process the bottom of the last floor is slightly reduced. The created building has to be rescaled to fit into its building lot.

relative to their containing blocks. Their approach generates parcels highly similar to those observed in real-word cities, but it mainly focuses on parcel layout and does not address all the phases of the city-generation process, unlike our method.

Aliaga et al. [9] presented a system for synthesizing urban landscapes by example. They proposed a random walk algorithm for obtaining parameters from existing cities that are later used in the generation process. Their system was somewhat capable of on-demand generation, but it was neither intended nor suitable for distributed environments because it required knowledge of all previously generated geometries for each future step.

On-demand world generation is highly related to texture synthesis algorithms. The main difference between these two approaches is the use of the generated results and whether the algorithm generates vector or raster output. Algorithms for texture synthesis usually use Voronoi diagrams [28] of randomly distributed points. One of the pioneering works in this area was published by Worley [125], who uses a function that complements Perlin fractal noise to produce textured surfaces resembling flagstone-like tiled areas, an organic crusty skin, crumpled paper, ice, rock, mountain ranges, and craters. Our algorithms are inspired by his function to determine the n^{th}-closest points that affect the structure of the texture at the currently generated area. We aim for a similar goal, but we use Delaunay triangulation instead. We also use techniques based on Voronoi diagrams to divide areas that are affected by different geometrical elements.

Liang et al. [67] presented another algorithm for synthesizing textures from an input sample in real-time, but they use a significantly different approach than ours and their results are not isotropic, which is important for stateless generation. Texture synthesis can also be used to generate street patterns [37, 110], but these works use different approaches that are difficult to adjust for the purposes of infinite cities. Lefebvre and Hoppe [64] presented an algorithm for parallel on-demand texture synthesis based on a neighborhood matching approach. Their scheme defines an infinite, deterministic, aperiodic texture from which rectangular views can be computed in real-time on a GPU. Another advance in infinite texture generation was made by Cohen et al. [26]. They utilized a small set of Wang Tiles to tile a plane non-periodically. Using a proper tile set, the texture can be extended on-demand. In 2007, Merrell [70] presented an algorithm for generating 3D buildings and cities from a set of 3D tiles. Merrell's later work [71] was focused on continuous city model synthesis. These techniques are, however, limited to structures aligned with the main axes.

To generate a realistic world structure, we must first analyze examples to acquire characteristics that are later used in procedural modeling. Important progress in inverse procedural modeling was made by Stava et al. [107]. They create parametric context-free L-systems that represent an input 2D model. Their approach is based on vector shape recognition, clustering in the transformation spaces and detecting structures as L-system rules. Elements and structures can be edited by changing the L-system parameters.

6.4 Chapter Conclusion

In our stateless-generation algorithm presented in the next chapter we aim to create a generator for infinite cities that would not be restricted to a regular rectangular grid. The key to this generator is a proper pseudo-random infinite generator of major roads. Because the major roads are at the beginning of the generating workflow, the rest of the city could be generated automatically using existing tools.

Our approach follows up on the related works above, combining their benefits and removing some of their limitations. Unlike [38, 39, 70, 71], our building lots can have various sizes and shapes, streets can be arbitrarily oriented, and the street network is not periodic. Unlike [9], our approach adds capabilities for distributed environments as well as the ability to generate only content related to the view frustum of the client.

Chapter 7

Stateless Generation of Distributed Worlds

7.1 Introduction

Much attention is currently focused on multi-user virtual environments hosted in the cloud for both gaming and non-gaming purposes [23, 45, 79]. With the arrival of massive multiplayer online games, game creators have had to deal with limited server capacity in terms of world size or number of players [21], but virtual-world services must be scalable [117]. Current cloud computing technologies are able to provide additional resources on-demand, but virtual-world systems are rarely able to generate on-demand game content (to save memory for world parts not needed at the moment). Another problem is the limited network connectivity of mobile clients in cellular networks, which is often too slow for downloading the content generated on the server, thus having a highly negative impact on the emerging and ubiquitous mobile gaming.

Our method is innovative in that it eliminates the need to synchronize the static content that was procedurally-generated on multiple devices. This will allow virtual-world servers to dedicate additional machines from the cloud environment to parallel content generation, or even to generate content on client devices. Because the content is generated on-demand, the virtual world can be considered theoretically infinite.

We refer to our method as *Stateless Generation* because it allows for locally generating only the content of the world that is relevant to the viewing frustum of the clients, and this content is generated independently, without knowledge of the states of the other generators.

As one of the most difficult virtual environments to generate, our efforts focus on generating an urban landscape (see Figure 7.1). City environments are generally complex

because they are structured and are both detailed and enormous. Procedural generation is a convenient tool for saving storage space and/or Internet bandwidth. When in the view frustum, buildings can be created on-demand from their lots (land parcels) using generating grammars.

Moreover, our method is general enough to be applicable to other types of structured landscapes (e.g., countryside, caves, labyrinths). Structured landscapes are generally more difficult to generate than unstructured landscapes (e.g., forests).

We provide a general purpose guideline for scalable algorithms generating virtual worlds that is applicable to most types of landscapes. We formalize the requirements and constraints such algorithms must fulfill. We then provide a novel algorithm for generating an infinite and scalable city-street layout, including a novel sub-algorithm for generating streets in a constrained environment, which could also be useful in traditional approaches to procedural city generation [114].

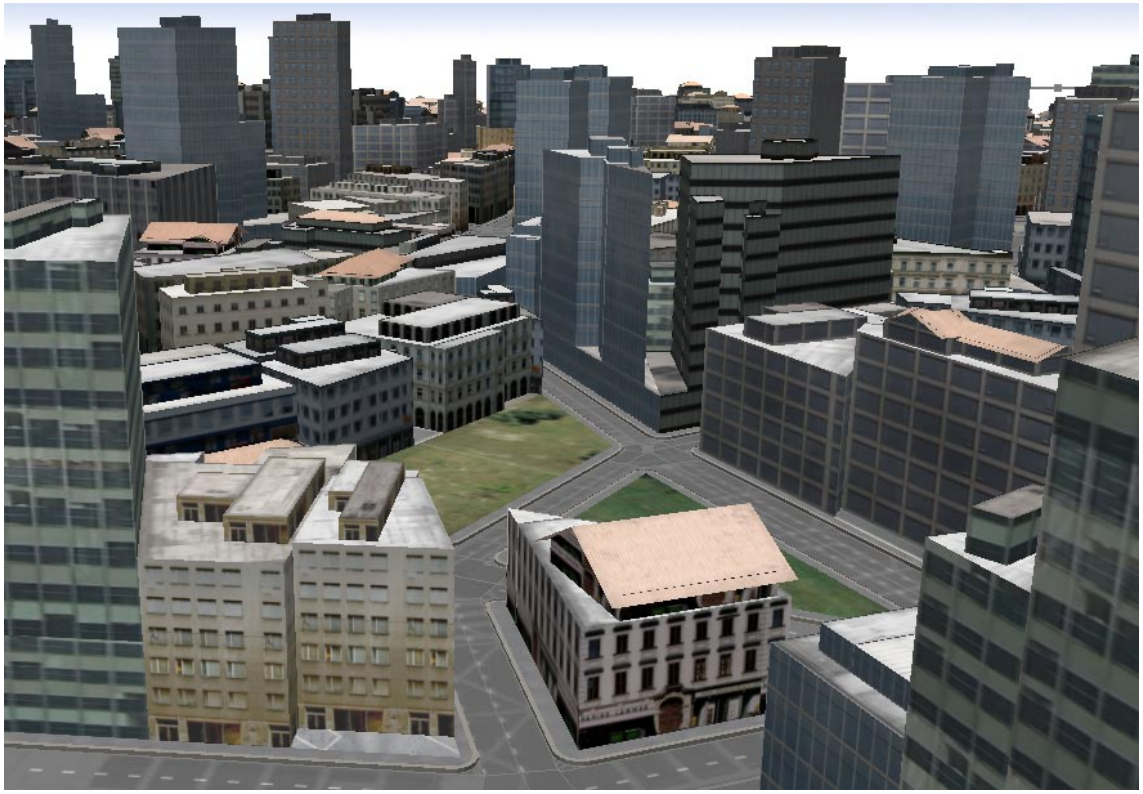

Figure 7.1: A stateless infinite city generated by our method

7.2 Stateless Generation Approach

We have laid down the following general requirements for our world generator:

1. The generated world is infinite and is not periodic.
2. Clients and/or servers are able to generate the static part of the world on-demand; they do not have to download it from a single (common) place. They should download only one random generator seed (or hash function) for the whole world.
3. Generation of the world can start from any point. The client will generate only those parts that are relevant (e.g., visible) to it.
4. The generation process is deterministic (usually achieved using pseudo-random generators). The results are always identical, regardless of the starting point or the area relevant to the client.

If a generator fulfills the above requirements, we call it a *stateless generator*. This is because its results do not depend on the results (or inner states) of other generators working in parallel or on its previously generated content. The defined requirements have the following outcomes:

a. Generated parts of the world can be cached. When any part of the world is cleared from the cache, it can be re-generated again, and it will be exactly the same as the original.

b. When multiple clients and/or servers generate the world, the intersected areas are exactly the same, and the generated virtual worlds connect seamlessly, despite that they started generating at different starting points and have not communicated with each other.

We assume that the world is infinite in two dimensions only and consists of buildings or other objects on an infinite plane. The 3D model is created later from the generated layout on the plane. We provide both a general purpose guideline and a specific version for generating the infinite urban landscape. The specific algorithm follows the standard city-generation workflow proposed by Parish and Müller [86], initially designed for finite cities. The workflow starts with the city street network, and buildings are generated afterward (phases: a. street network, b. building blocks/lots, c. building geometry). We explain our approach to city street-network generation, while the other phases remain nearly the same as in the case of finite cities. When working with infinite structures, we assume that we will compute only those values that are needed due to the intersection with the view frustum.

Street-network characteristics

To achieve the desired city appearance, our algorithms use parameters that can either be provided by the user or measured from an example of a city map. We will denote these parameters as street-network characteristics.

When processing a map of an existing city, streets are replaced by straight segments between street junctions. These segments are called street segments. Street segments are also created during the generation process, and they are converted to streets in the final phase of the algorithm.

To acquire the street-network characteristics from a map, we use the random walk algorithm as described by Aliaga et al. [9] applied to city-map street segments. A detailed description of the street network characteristics and the corresponding terminology we adopt can also be found there. For our purposes, the street network characteristics contain the following parameters:

1. average length of a street segment and its variance
2. average unoriented angle between two consecutive street segments and its variance
3. average number of street junctions on a surface unit (e.g., a square kilometer)
4. average number of street segments on a surface unit
5. street tortuosity – the average ratio between street length and the corresponding street segment length

Our algorithms make an effort to ensure that the generated street network will have similar network characteristics to those of the original city map, but currently the resulting distribution exactly corresponds only in mean values.

7.3 General Algorithm for Stateless Infinite Worlds

This section describes general guidelines for designing algorithms for stateless infinite worlds. Later, we show how we apply this algorithm template to urban landscapes.

The algorithm input consists of items shared among generators (in the case of multiple generators): the generator parameters (e.g., street-network characteristics); the tessellation grid parameter d - as described below; and the hash function for the pseudo random generator. Other input parameters are individual per each generator: the position, orientation and field-of-view of the camera in the infinite world, with near and far clipping plane distance (this is usually called the *view frustum*).

The algorithm output is the geometry of objects within the view frustum. Note that because the algorithm works in a 2D plane, we work with view frustum projections to the plane rather than the corresponding 3D representation. The shape is not important for the algorithm; it works with any bounded 2D area.

Each step of the algorithm is now described in greater detail below:

Step 1 – Tessellation:
Create a non-periodic tessellation of the infinite plane. The tessellation is only an auxiliary structure not visible in the final model. Subdividing the plane into a set of bounded areas (called tessellation fragments) allows us to transform the problem into sub-problems of generating content only in the areas intersected with the view frustum. In Section 7.4, we provide a novel sub-algorithm for the tessellation of an infinite plane into a non-periodical set of arbitrarily oriented triangles. We prove that this algorithm fulfills the requirements for Stateless Generation.

Step 2 – Generation of interfaces:
Use a pseudo-random generator to define interfaces (objects that cross the fragment boundary) between the adjacent tessellation fragments. In Section 7.5, we provide a sub-algorithm that generates a street layout for an urban landscape, where the interfaces are the street-segments that intersect tessellation fragment boundaries.

Step 3 – Constrained generation:
Generate the inner part of the fragments while preserving the constraints set by the interfaces. In Section 7.6, we provide another novel sub-algorithm for generating a street network in a constrained environment that solves this problem for street layouts.

Step 4 – Procedural generation:
Use traditional procedural generation to finish the landscape. This is the only step performed in 3D.

7.4 Tessellation Algorithm

This algorithm divides the infinite plane into a tessellation of triangular tessellation fragments. It provides only those fragments that are fully or partially within the client's view frustum.

The algorithm input parameters are the view frustum; a hash function; and the grid size parameter d. As its output, the tessellation fragments that intersect the frustum are

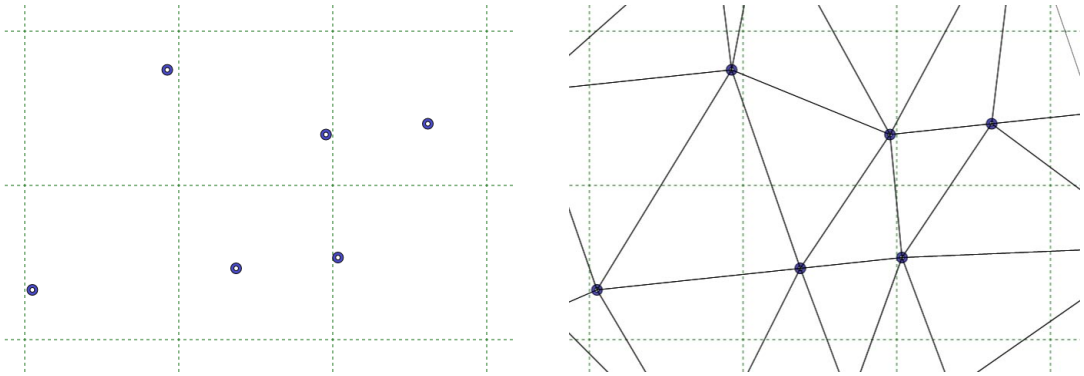

Figure 7.2: **Tessellation algorithm visualization** – Left: An infinite tessellation grid (step 1) with pseudo-randomly generated tessellation points (step 2); Right: Infinite Delaunay triangulation that forms tessellation fragments (step 3); These structures are only auxiliary for further phases and do not represent the final city geometry.

provided. Note that the geometry of the fragments remains consistent in the case of multiple overlapping queries.

First, we present a theoretical approach that operates with infinite structures. Then, we provide practical instructions about how to correctly generate the content within the view frustum using limited computational resources. We achieve this by omitting data that do not impact the content of the view frustum. Note that, due to the connected structure of the world, even objects near but outside the view frustum can affect objects inside the view frustum; we have to take this into account. The approach is similar to the lazy-evaluation method used in functional programming languages (e.g., Haskell) to work with infinite data structures.

Theoretical approach

Theoretical step 1:

Suppose we have a plane dedicated for generating the world. Create a regular infinite square grid on the plane. (The segment length of the grid should be a configurable generator parameter. Let us denote the length as d.) We denote the grid as the tessellation grid. The grid will divide the infinite plane space into squares. Each of these squares has an X and Y integer coordinate (positive or negative). Apply a hash function to each pair of coordinates to acquire a pseudo-random generator seed for each square (see Figure 7.2). We use universal hashing functions for integer vectors of size 2.

Theoretical step 2:

Use the seed to generate a pseudo-random position inside the given square (with

uniform distribution) and place a node on that position. We will refer to these nodes as tessellation nodes.

Theoretical step 3:

Create a Delaunay triangulation using the tessellation nodes. All tessellation nodes are connected with triangulation edges to form an infinitely large tessellation. In Section 7.4, we prove that Delaunay triangulation always produces the same results, regardless of its starting area, and can be performed partially only using a finite subset of tessellation points for any bounded area. We use the triangulation technique described below to find the proper edges.

Triangulation

We must select a robust and unequivocal triangulation method that requires knowledge only of the local surroundings of the processed nodes, as an infinite number of nodes cannot fit into memory. Delaunay triangulation has these desirable properties for Stateless Generation.

Definition of Delaunay triangulation:

Three nodes form a triangle that belongs to Delaunay triangulation if and only if there are no other nodes inside its circumcircle.

We will now prove that Delaunay triangulation has properties of Stateless Generation. We will start with several lemmas.

Lemma 1:

In a tessellation grid, any circle with a radius greater than $\sqrt{2}d$ has at least one tessellation node inside it.

Proof:

Such a circle contains at least one whole grid square inside. Therefore, it also contains a tessellation node corresponding to that square (see Figure 7.3).

Lemma 2:

In a tessellation grid with Delaunay triangulation performed on its tessellation nodes, no triangulation edge may be longer than $2\sqrt{2}d$.

Proof:

According to the definition of Delaunay triangulation and lemma 1, every triangulation edge has to be part of a triangle with a circumcircle radius smaller than $\sqrt{2}d$. Edges greater than $2\sqrt{2}d$ will not fit into the circumcircles.

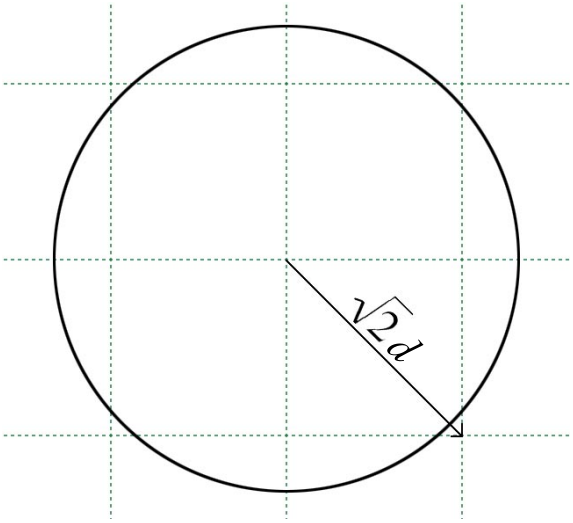

Figure 7.3: The circle has a radius slightly less than $\sqrt{2}d$ and does not contain any of the squares of the infinite grid. It is obvious that every circle with a radius greater than $\sqrt{2}d$ contains at least one whole square and its tessellation node.

Lemma 3:

If we remove some nodes from a Delaunay triangulation and perform a new Delaunay triangulation on the reduced set of nodes, all triangles from the original triangulation that were not using any of the removed nodes will be present in the new triangulation. (Some new triangles may emerge, but that is fine.)

Proof:

It is an outcome of the definition of Delaunay triangulation. (No new nodes will appear inside the original circumcircles.)

Lemma 4:

When we have Delaunay triangulation performed on tessellation nodes of a tessellation grid and a bounded area, removing the nodes that are farther from the area than $2\sqrt{2}d$ will have no effect on the triangles that intersect with the area.

Proof:

It is a direct outcome of lemma 2 and 3.

Theorem:

The Delaunay triangulation performed on tessellation nodes of a tessellation grid fulfills the definition of Stateless Generation described in Section 7.2.

Proof:

We will follow the definition from Section 7.2:

1. The structure is infinite and is not periodic.
2. The triangulation can be generated by multiple generators independently.
3. Parts not relevant to a bounded area can be removed using lemma 4.
4. The process is deterministic. □

Delaunay triangulation is unique except for situations where there are 4 nodes on a circle. This, however, happens with negligible probability for randomly generated points. We use computation with particular high-precision real numbers, which substantially decreases the probability of such cases and provides the same results on different operating systems and CPUs.

Practical approach

This approach describes how to generate the part of the infinite tessellation that is within the view frustum.

Step 1:

Take the view frustum and enlarge it by $2\sqrt{2}d$ in all directions. (Due to lemma 4 from the previous section, it is guaranteed that triangulation inside the view frustum will not be affected by content outside the enlarged view frustum.)

Step 2:

Take all squares that intersect with the enlarged view frustum and generate their corresponding tessellation nodes. Filter out the tessellation nodes that are outside the extended frustum. We now have all the tessellation nodes that are within the extended view frustum.

Step 3:

Create Delaunay triangulation on those nodes. Lemma 4 guarantees that we generate all triangles that intersect the view frustum correctly. Filter out triangles that are completely outside the original frustum (these may not be generated correctly). We have now acquired the part of the tessellation that is visible in the view frustum and conforms to the Stateless Generation properties.

Complexity:

The most complex step of the algorithm is the triangulation. The well-known DeWall algorithm [24] creates a Delaunay triangulation for n nodes in $O(n \log \log n)$ time. However, we use the previous incremental algorithm for Delaunay triangulation [62],

with time complexity $O(n \log n)$, which allows us to include additional nodes in the triangulation once the initial set is finished. This is a nice feature in cases when the user and his or her view frustum move and thus the visible area changes over time. Each additional node can be added in $O(\log n)$ time.

7.5 Generating the Tessellation Fragment Interfaces

In the case of an urban landscape, the interfaces consist of street segments that cross the tessellation fragment boundary. These street segments form constraints that guarantee a continuous street network between the tessellation fragments.

The generation takes as its *Input* the processed tessellation fragment together with the tessellation fragments adjacent to it, plus the street-network characteristics. Its *Output* is the street segments that cross the boundary. The street segments do not cross each other and each of them intersects the target tessellation fragment and exactly one other tessellation fragment.

First, we must create a seed for the pseudo-random generator to generate random numbers consistently. We create one pseudo-random generator for each boundary edge. To do this, we take the coordinates of the grid squares in which the tessellation nodes were created (4 integers from two squares related to the edge endpoints) and apply a hash function to them to find the seed. We use the hashing function

$$h = [c_1(x_1 + x_2) + c_2(y_1 + y_2)] \mod c_3$$

where c_1, c_2, c_3 are random large prime number constants. This ensures that the results are the same when we start generating from the other tessellation fragment and that the transition of the street network is seamless.

For each of the boundary edges we use the Poisson distribution to determine the number of street segments that cross the edge. The distribution mean is set proportionally to the product of the boundary-edge length, of the average street-segment length and of the average number of street segments on a unit surface to reflect the original network characteristics. For each street segment, we generate a point on the boundary edge where the street segment crosses the edge (uniform distribution), the length of the street segment (normal distribution according to the average street segment-length and variance), the portion of the street segment that will be in the target fragment (uniform distribution) and the angle between the street segment and the edge (the distribution is proportional to the sine of the angle). See Figure 7.4 for examples.

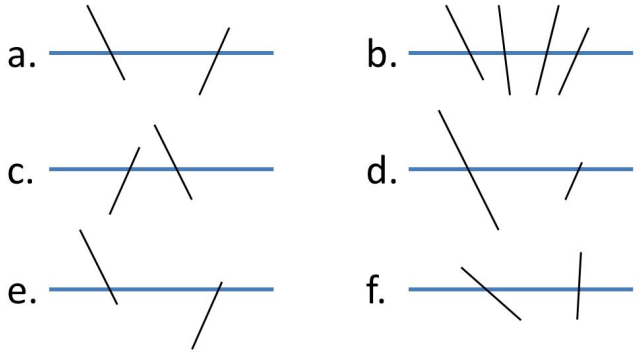

Figure 7.4: **Variance in street segment interface generation** – (a) reference case, (b) different numbers of street segments, (c) different crossing points, (d) different lengths, (e) different portion of street segments in the target fragment, (f) different angles

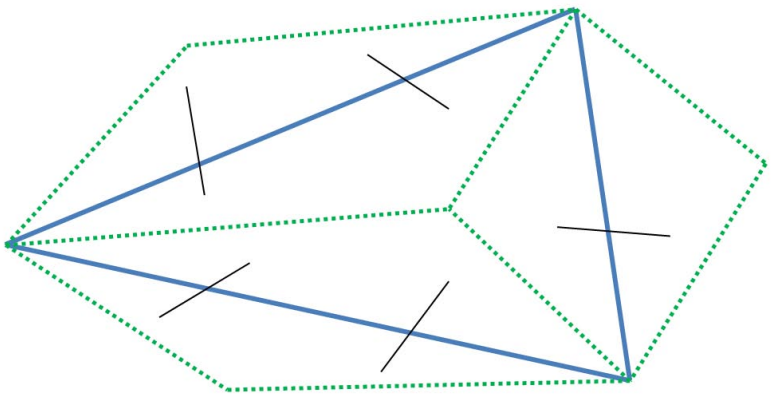

Figure 7.5: **A tessellation fragment (blue) and its generated interface street segments (black)**; The endpoints of the street segments must be closer to its boundary edge, i.e., they must be within the dotted green boundary.

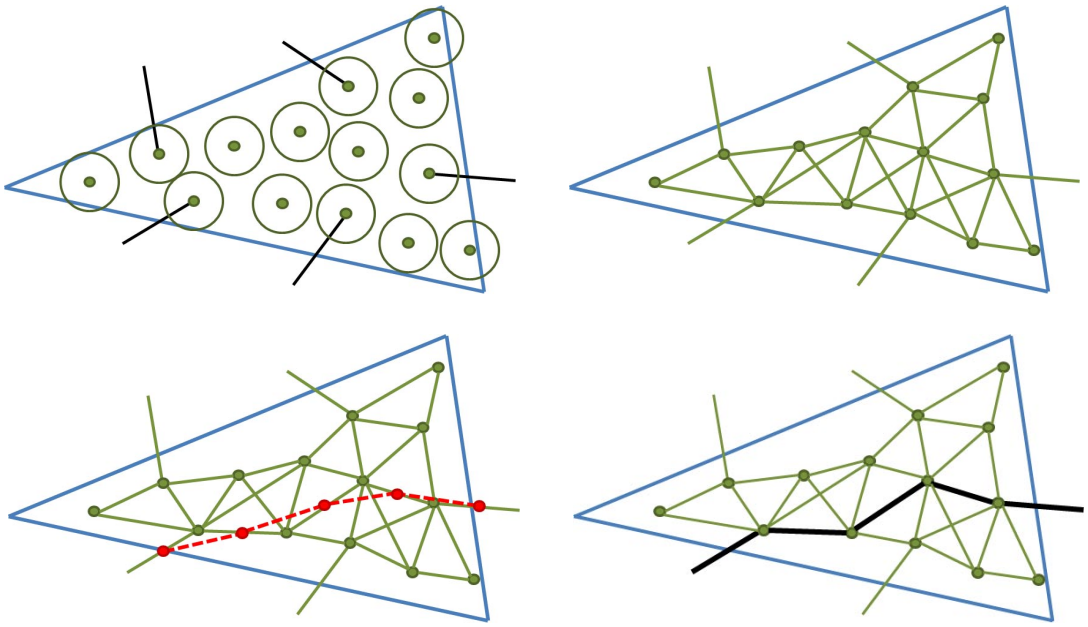

Figure 7.6: **Constrained street-generation algorithm visualization** – Top-left: A tessellation fragment (blue) with street network interfaces (black) during the phase of generating junction candidates using Poisson disks Top-right: Street network segment candidates graph (green) Bottom-left: The found dual path P^L (red, dashed) with minimum evaluation in the line graph. Bottom-right: Path P (black) added to the street network; crossing street-network candidates have been removed.

Each of the generated street segments is then tested for the following conditions:

1. The street segment does not cross any previously generated street segment for this boundary edge.
2. The endpoints of the street segment are closer to its boundary edge than to any other boundary edge. (see Figure 7.5)

If the street segment does not fulfill the conditions above, we discard it and generate a new one instead.

7.6 Constrained Street-generation Algorithm

This algorithm generates the interior of the tessellation fragment with respect to the constraints set by interfaces that consist of street segments. Currently, the algorithm can be used only for a street network with no superimposed pattern. The algorithm will not produce rectangular street networks (New York, Chicago) or radial to center street networks (Paris). Those and other types of street network patterns are described in [86].

It is possible to create another Stateless Generation algorithm for each of these patterns, but it seems to be difficult to find a universal algorithm for all of them due to the already fixed positions of the tessellation fragment interfaces generated in the previous phase.

The algorithm *Input* parameters are the following: a convex shape (i.e., the tessellation fragment) with street segments crossing its boundaries (from the previous phase); and a pseudo-random generator (from the coordinates of the tessellation nodes of the triangle). The algorithm works generally for any convex shape, although in practice we use triangular shapes generated by the previous tessellation algorithm. Its *Output* is the generated street network inside the shape that takes the interfaces into account.

Each step of the algorithm is shown in Figure 7.6. First, we use the pseudo-random generator to place nodes that will be junction candidates. We use Poisson disk distribution for this task, and for simplification, the radius is equal to one-fourth the average street-segment length from the street-network characteristics. The end points of the interface street segments are also junction candidates. Next, we connect each pair of junction candidates with an edge (which we call a street-segment candidate) except for the following: (i) those that are too improbable according to the average street-segment length and its variance in the street-network characteristics (exceeding a predefined threshold) and (ii) those that would cross the interface street segments. We then add the interface street segments to the set of street-segment candidates. To generate the street network, we incrementally add the paths of street-segment candidates into the street network. We try to add paths that are in accordance with the street network characteristics. To do this step, we create an evaluation function to determine the appropriate paths to be added. Using this function, we transform our problem to finding the shortest path in a weighted graph.

Street-segment path evaluation

We now must define a set of evaluation functions f for street segments, angles and paths. For realistic street generation, we have to create functions that assign a low value to paths according to the street-network characteristics. For street segments ($e \in E$) and angles between two adjacent street segments (α), we have defined the evaluation functions as follows:

$$f(e) = \frac{1}{f_x(length(e))} \tag{7.1}$$

$$f(\alpha) = \frac{1}{f_x(\alpha)} \tag{7.2}$$

where f_x is the probability density from the street network characteristics (normal distribution based on the average length/angle and its variance)

For paths, we have defined the path evaluation function $f(P)$ as follows: Let $P = (e_1, e_2, ..., e_n)$ be a path, then its evaluation is defined as:

$$f(P) = B(e_1)B(e_n)[f(e_1) + f(\alpha_{1,2}) + f(e_2) + f(\alpha_{2,3}) + \cdots + f(\alpha_{n-1,n}) + f(e_n)] \quad (7.3)$$

where $B(e)$ is a penalty for path-ending street segments based on the degree of the end nodes (i.e., junctions) of the path (the number of incident street segments already added to the street network). Note the difference between added street segments and street-segment candidates.

$$\begin{aligned} B(e) &= 10 && \text{for } degree = 0 \quad \text{(i.e. dead end)} \\ B(e) &= 1 && \text{if } e \text{ is an interface street segment} \\ B(e) &= degree && \text{for } degree > 0, \text{ not an interface} \end{aligned}$$

The purpose of the penalty function is to prefer paths that connect street-segment interfaces and to reduce the number of paths with dead ends. The values of the penalty function are empirical.

Our goal is to find a path in G with the minimum sum of evaluation values on its edges and angles. Because the common path-finding algorithms do not work for values on angles, we transform the original graph G to its dual form, called the line graph $L(G)$. This transforms angles to edges. After we find the evaluation for paths in $L(G)$ using a standard pathfinding algorithm, we perform a reverse transformation to obtain the path evaluations in graph G.

The line graph (also called the edge-to-vertex dual graph) $L(G)$ represents the adjacencies between the edges of G, and it has the following properties:

1. Each node of $L(G)$ represents an edge of G.

2. Two nodes of $L(G)$ have a common edge if and only if their corresponding edges in G share a common node.

Let us define the evaluation function f for the edges in the line graph:
Let $e^L_{a,b}$ be an edge in $L(G)$ corresponding to a pair of adjacent edges e_a and e_b in G.
Let $\alpha_{a,b}$ be an angle between the edges e_a and e_b

Let us define the evaluation function for the edge in the line graph $f(e^L_{a,b})$ as:

$$f(e^L_{a,b}) = \frac{1}{2}f(e_a) + f(\alpha_{a,b}) + \frac{1}{2}f(e_b) \tag{7.4}$$

Let $P^L = (e^L_{1,2}, e^L_{2,3}, \ldots, e^L_{n-1,n})$ be a path in the line graph; we define $f(P^L)$ as:

$$f(P^L) = f(e^L_{1,2}) + f(e^L_{2,3}) + \cdots + f(e^L_{n-1,n}) \tag{7.5}$$

Let us substitute $e^L_{a,b}$ using formula 7.4:

$$\begin{aligned} f(P^L) \;=\; & \left(\frac{1}{2}f(e_1) + f(\alpha_{1,2}) + \frac{1}{2}f(e_2)\right) + \\ & \left(\frac{1}{2}f(e_2) + f(\alpha_{2,3}) + \frac{1}{2}f(e_3)\right) + \cdots + \\ & \left(\frac{1}{2}f(e_{n-1}) + f(\alpha_{n-1,n}) + \frac{1}{2}f(e_n)\right) \end{aligned} \tag{7.6}$$

We simplify the expression to:

$$\begin{gathered} f(P^L) = \tfrac{1}{2}f(e_1) + f(\alpha_{1,2}) + f(e_2) + f(\alpha_{2,3}) + \\ f(e_3) + \cdots + f(e_{n-1}) + f(\alpha_{n-1,n}) + \tfrac{1}{2}f(e_n) \end{gathered}$$

We use the substitution from formula 7.3:

$$f(P^L) = \frac{f(P)}{B(e_1)B(e_n)} - \frac{1}{2}\left(f(e_1) + f(e_n)\right) \tag{7.7}$$

We express $f(P)$ from the previous formula:

$$f(P) = \left(f(P^L) + \frac{f(e_1) + f(e_n)}{2}\right) B(e_1)B(e_n) \tag{7.8}$$

Minimum evaluation path-adding algorithm

1. From a given graph G, create its line graph $L(G)$.

2. Using the Floyd-Warshall algorithm, find the path with the smallest evaluation $f(P^L)$ between each pair of nodes in $L(G)$.

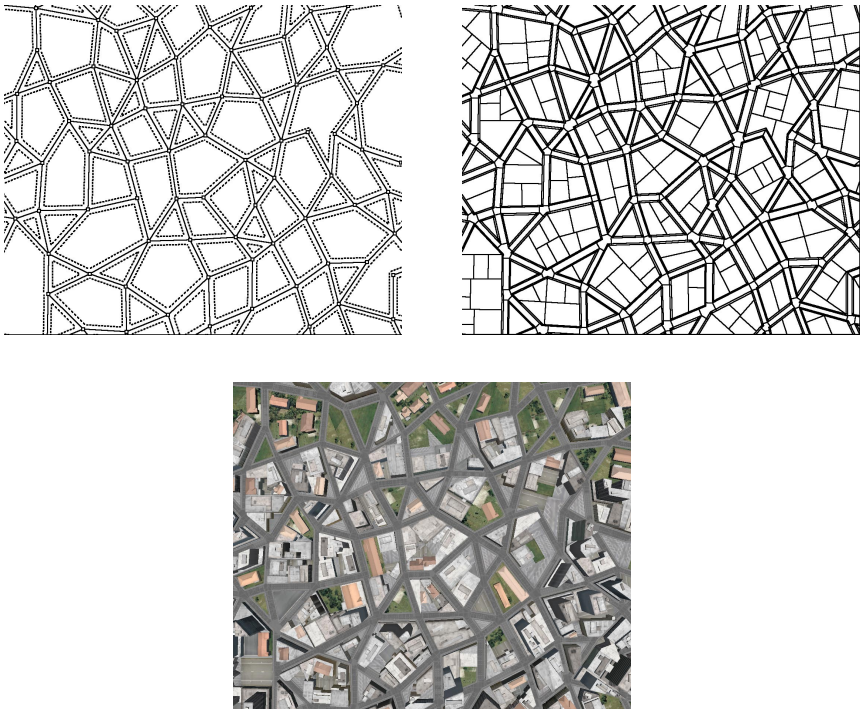

Figure 7.7: Top left: Generated street network from top view; Top right: Added building lots; Bottom: Added building geometry

3. Using formula 7.8, compute $f(P)$ for paths dual to the paths from the previous step. (Note that dual paths have a minimal evaluation in G between the appropriate nodes as an outcome of formula 7.8. Also, note that $f(e_1)$, $f(e_n)$, $B(e_1)$ and $B(e_n)$ are constant for all paths between edges e_1 and e_n.)

4. Find the path in G with the minimum evaluation.

5. Add the path to the street network.

6. Remove all street segment candidates that cross the added path.

7. Repeat steps 1 – 6 until either the average number of street segments on a surface unit in the processed fragment is higher than the number from the street network characteristics, or no street candidates are left.

8. Add all remaining street network interfaces to the street network if they were not added by previous steps (this is quite rare because street network interfaces are preferred by the penalty function.)

The described path-adding algorithm presents an issue we must address. It is not guaranteed that the path in G with the minimum evaluation value will not cross itself,

which is a situation we want to avoid. This happens quite rarely because self-crossing paths usually have a high evaluation value, but it is still possible. For this reason, we check the selected path for self-crossing. If self-crossing is found, we do not add the street segment that crosses a previously added street segment in step 5 and any subsequent segment of the processed path.

Complexity:

Because of the Poisson disk distribution of the junction candidates, the number of street-segment candidates is linear to the number of junction candidates $O(n)$. The number of edges and vertices of the line graph $L(G)$ is also $O(n)$. The Floyd-Warshall algorithm requires $O(n^3)$ time to compute the evaluation for all possible paths. The path evaluation has to be re-computed in each cycle, so the total time complexity is $O(n^4)$ in the worst case. The average case is faster because we add more than one street-segment candidate with an average path.

Optional post-processing:

When the process of generating street segments is finished, we generate the rest of the city using a combination of existing algorithms. As the first step, we can optionally convert the straight street segments into curved streets. This step is not needed when we work with short average street segments. We do this using the well-known midpoint displacements algorithm based on the measured tortuosity. To guarantee that the curved streets will not intersect, we create a Voronoi diagram for the original street segments and re-generate all displacement operations that are outside the bounds of the corresponding Voronoi cell until the displacement is within the cell.

7.7 Generating Building Lots and Geometry

When the street network is complete, we create building blocks out of the areas bounded by streets. In the next step, we subdivide them into building lots, using an algorithm described by Parish and Müller [86]. Figure 7.7 shows different phases of the modeling process. The subdivision algorithm described by Aliaga et al. [9] would also be suitable.

To generate the building geometry from the corresponding lots, we use an approach based on L-systems [92] that was extended by Wonka et al. [124] and later significantly improved by Müller et al. [76]. This approach generates a building geometry using CGA (Computer Generated Architecture) grammar that could be created by a model designer or obtained from existing buildings using a semi-automatic process proposed in [8]. We use

an existing grammar that is provided with the CityEngine [32] modeling software. One CGA grammar can generate many building variations.

7.8 Limitations and Potential Extension

This section discusses the limitations of our approach and offers suggestions on how they could be overcome. We also discuss how to combine our algorithms with previous related work. Our current implementation does not contain these techniques.

The street generation algorithm presented here does not currently support multiple street types (e.g., highways, main roads and narrow streets), but it can be extended to support them. First, we need to acquire separate street-network characteristics for each type of road. Then, we create street-interfaces for all types of streets. The constraint generation phase is performed multiple times for each street type, from the widest to the narrowest.

Our current implementation also does not take into account space-correlations of individual building types (e.g., industrial buildings are often placed together in existing cities, as are skyscrapers). This can be solved by creating an additional overlapping infinite quarter-type structure (e.g., a grid or a different pseudo-randomly generated Delaunay triangulation) that will provide building-type-probability parameters for the used CGA grammar. The quarter type would be determined by a hash function for each cell of the structure.

Currently, we are working with cities on a flat terrain only. However, our approach can be combined with existing techniques for procedural terrain generation, e.g. [15], to create more realistic infinite cities. The corresponding street-generation algorithm also has to be altered, e.g., to prefer streets on a flat terrain rather than on a bumpy one.

Procedurally generated architectures using our approach can be combined with a finite number of manually designed buildings. We suppose that the designed content is surrounded by a street loop. In such a case, we add all the endpoints of the street segments of the street loop to the pool of junction candidates when we perform the constrained street-generation algorithm of the corresponding Delaunay triangle. In the later phase, we choose the segments of the street loop first, and we do not perform an evaluation for them. We can proceed similarly when the designed content crosses the border of two or more Delaunay triangles.

By substituting the constrained generation algorithm, one can generate different types of worlds than cities, such as mazes, building interiors or electric circuits. Every new type of content will require certain minor modifications.

Moreover, we have identified certain weaknesses of the algorithm that we are unable to overcome at the moment. Our method does not provide good results when the input street-network characteristics have been obtained from an existing map with diverse types of street layouts, e.g., city center and surrounding suburbs. In such cases, the resulting street layout does not reflect either of the original structures. Another problem arises when distances between street crossings are relatively long compared to the tessellation fragments - this can occur particularly in the case of highways. In such cases, the constrained street-generation algorithm does not produce adequate results. We have also noticed performance problems in the case of high-density street layouts with large tessellation fragments due to the $O(n^4)$ complexity of the algorithm. We therefore recommend the use of smaller tessellation fragments for dense street layouts with short streets and greater ones for a sparse layout with long streets. Our method is also limited to simple CGA grammars that can be generated in real time and that provide a reasonably low number of polygons per building (up to 50). The sizes of the generated lots must also reflect the possibilities of the CGA grammar — some grammars can have a limit on the maximum or minimum lot sizes. Fortunately, most grammars can overcome this limitation by generating a simple standby geometry, usually an empty lot or a parking space.

7.9 Applications

We have found two main scenarios that perfectly take advantage of our approach:

The first scenario uses a distributed 3D world on multiple client devices with sufficient computation power but with limited network throughput. This reflects current mobile phones, tablets and laptops in areas without a high-speed mobile connection or Wi-Fi connectivity. The devices would use a simple server infrastructure to share the hash function and would then be able to generate the static content of the world on their own. They only need to synchronize dynamic changes and positions of users in the world.

The second scenario uses servers in the cloud environment for generating the world and providing the generated content to the connected clients. The Stateless Generation property allows us to dynamically add more instances of the server on-demand based on the number of clients connected and on the area that needs to be covered. The servers can generate content independently, and they do not have to synchronize their work. Nevertheless, having a common cache for already generated content would be beneficial to them. This scenario is useful for a demanding world-generation process and for clients with limited computational power (e.g., low-end mobile phones, streaming to TV). This can also be used in computer games to prevent players from cheating by preventing the client from

accessing data it does not need to display. Content is generated in the cloud and is then sent to the client devices as vector geometry or is streamed as video. In the case of vector geometry, the client device caches received content to save bandwidth for future queries.

7.10 Implementation

To verify that our approach is suitable on multiple platforms, we have developed mobile, laptop/tablet, and web browser applications that interactively generate infinite street networks according to the generator parameters. Our implementation follows the first scenario from the previous section. We did not implement the second scenario.

The mobile application runs on the Windows Phone 8 operating system using WinPRT technology. The laptop/tablet application uses WinRT technology and can be used on Windows 8 laptops and tablets (e.g., Microsoft Surface), including the limited Windows RT operating system. The web application uses Silverlight 5 technology [74] (.Net equivalent of a Java applet). All our implementations are written in C# and share common parts of the code using C# Portable Libraries. The graphics are rendered using the cross-platform XNA library [68, 90].

We have considered using existing software for geometry generation from CGA grammars to avoid re-implementing them. However, none of the common procedural building modeling tools currently support on-demand generation controlled by 3rd party programs. We therefore created an automatic export module for the CityEngine for on-demand generation using its Jython scripting (Python based on Java Runtime), although plugins are not officially supported by this software. According to CityEngine customer support, future versions should contain official support for plugins. The plugin allows us to generate buildings on demand from their lot shapes. This approach enabled us to generate the city without re-implementing the CGA grammar interpreter. In real scenarios, it is not suitable to include CityEngine in the client application because of its licensing policy. CityEngine cannot be executed on mobile phones, so we run it remotely on a separate server in that case. For the requirements of real future computer games and simulations, we assume that the CGA grammar interpreter will be implemented as a sub-program of the client application. CityEngine plugin details are shown in Figure 7.8.

Because different platforms provide different implementations of their standard pseudo-random generators, we provide our own unified implementation of the pseudo-random generators and the hash function that is based on Donald E. Knuth's subtractive random number generator algorithm [54].

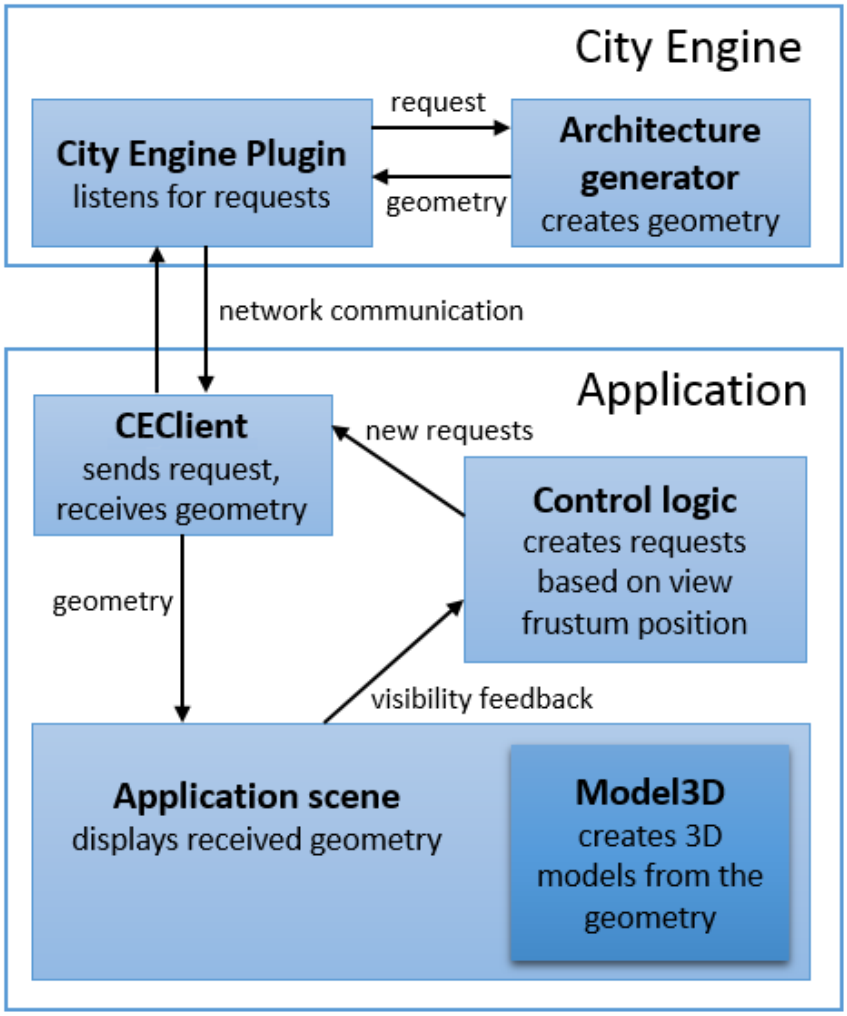

Figure 7.8: **CityEngine plugin scheme**; The plugin can connect to the CityEngine application from a remote station and take advantage of the CityEngine tools from distance in real time. The user can also remotely set grammar rules for desired building models. For the implementation of the plugin we chose Jython scripting language that is integrated in the CityEngine. Our solution consists of the server part written in Jython that controls the modeling software and the client part that can query the server part for building geometry. The server part is capable to receive grammar rules for generating of geometry and a shape of the initial lot for the requested building. Then the server part returns the desired building geometry to the client part. We would like to thank Jaroslav Minařík for help with the implementation of the CityEngine plugin.

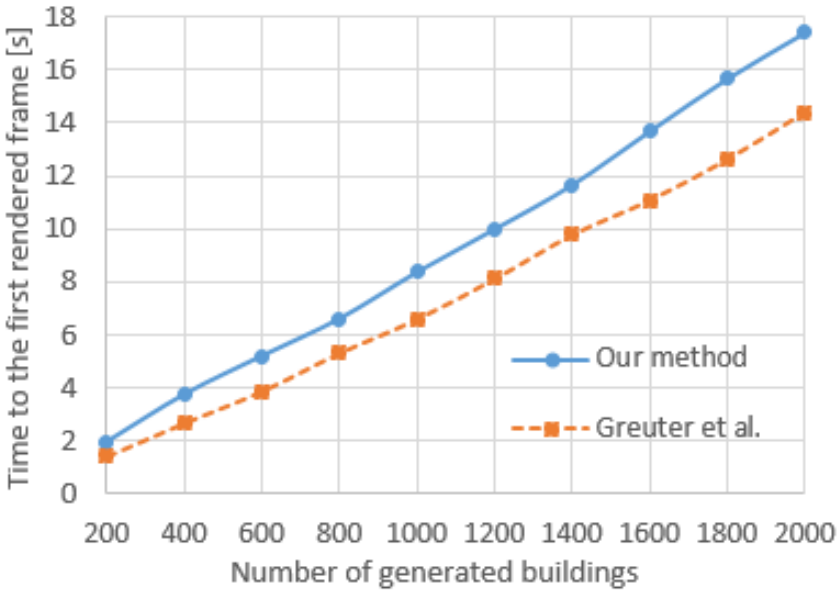

Figure 7.9: Comparison of the performance of methods for city-layout generation (average from 10 measurements with different seeds). Our method is more demanding.

7.11 Performance and Measurements

We measured the performance of the application on a Sony Vaio S15 laptop (Windows 8; processor: Intel Core i7-3612QM Quad-core 2.1 GHz; 8 GB RAM; graphics card: Intel HD 4000; resolution: 1920×1080). First, we measured the time until the first frame is rendered. This requires that all buildings in the view frustum be generated beforehand. We compare our method with the previous method published by Greuter et al. [38]. Their approach creates many more streets and fewer buildings per surface unit. To compensate for this , we used different view frustum sizes for each of the methods to achieve roughly the same number of buildings in the view frustum. We performed the measurements for multiple frustum sizes that are in accordance with multiple numbers of generated buildings, see Figure 7.9.

Our method is slightly slower, as it generates more advanced streets, but both methods run at approximately the same speed. Next, we measured the number of frames per second (FPS) for a static camera and a moving camera. The moving view frustum case has to address on-demand generation of the street network and additional buildings. Figure 7.10 shows how a walking or running user fits into the real-time frame-rate rendering speed. The interactive frame-rate (above 5 FPS) is still maintained for a higher speed.

To prove that our approach can be used on mobile phones, we performed measurements on a Nokia Lumia 920 smartphone (Windows Phone 8, Qualcomm MSM8960 Snapdragon

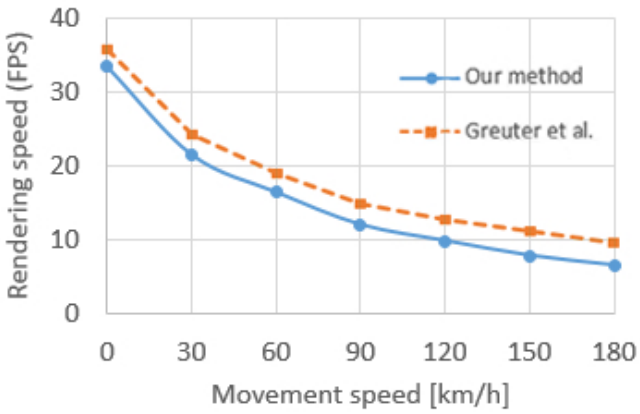

Figure 7.10: **Rendering speed for a moving user**; the content is generated on-demand (average from 10 measurements with different seeds). The experiment was performed with a view frustum that contains approximately 1000 simultaneously displayed buildings. The average building in the measurement has 35.4 textured polygons.

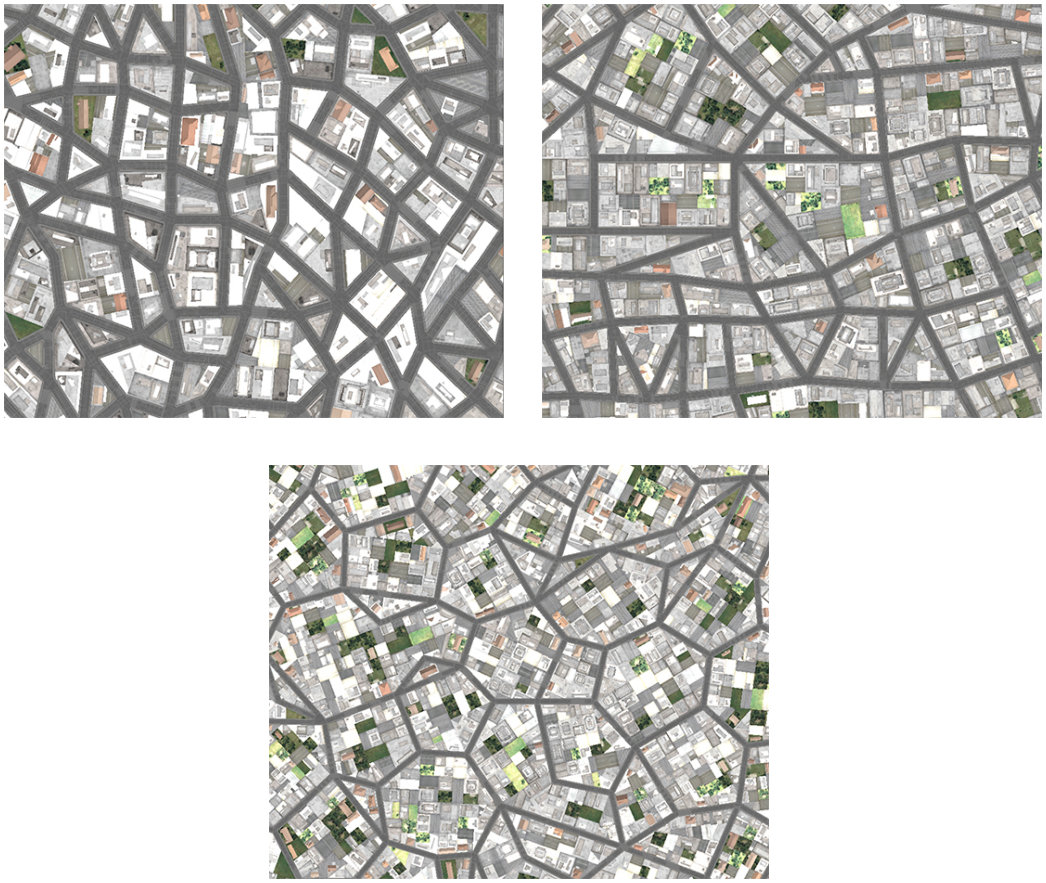

Figure 7.11: **Examples of street network varieties** - Left: A dense street network based on triangular street shapes; Middle: A medium-dense network based on rectangular street shapes; Right: A low-density street network based on hexagonal shapes; Note that the results do not perfectly reflect the geometrical shapes of the original network.

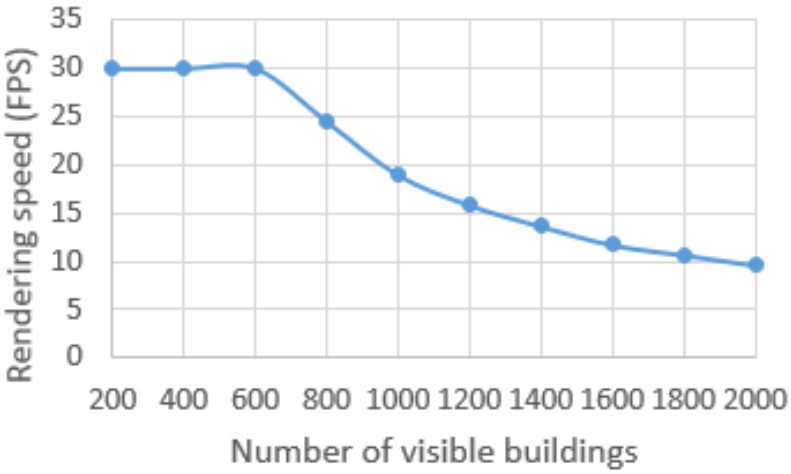

Figure 7.12: Rendering speed for different numbers of buildings on a Nokia Lumia 920. The phone refresh limit is 30 FPS.

Dual-core 1.5 GHz processor, 1 GB RAM, Adreno 225 graphics card, resolution 1280×768). In this case, the mobile phone is used to render a geometry that is generated on a remote computer. The results are shown in Figure 7.12. The phone is capable of rendering up to 800 buildings in real time. In this test, we do not consider any delay caused by data transfer from a remote computer. These issues are discussed in [A.3].

7.12 Chapter Conclusions

We have developed an algorithm for generating a possibly infinite street network on-demand. The main advantage of the algorithm is that it can generate only the content that is visible to the client and that the generated content is consistent in the case of multiple clients. Our appearance of an infinite city looks more natural than that of the previous approach developed by Geuter et al. (compare the results in figures 7.1 and 6.5), although it does not perfectly simulate the patterns of the example of existing street network, as the path-selection heuristics do not necessarily result in a perfectly corresponding distribution of segment lengths - see examples in Figure 7.11. Additional examples of our method are shown in Figure 7.13. The algorithm can be used in real-time both on personal computers and on mobile phones. We have also defined requirements and general guidelines for Stateless Generation that can be used for other algorithms.

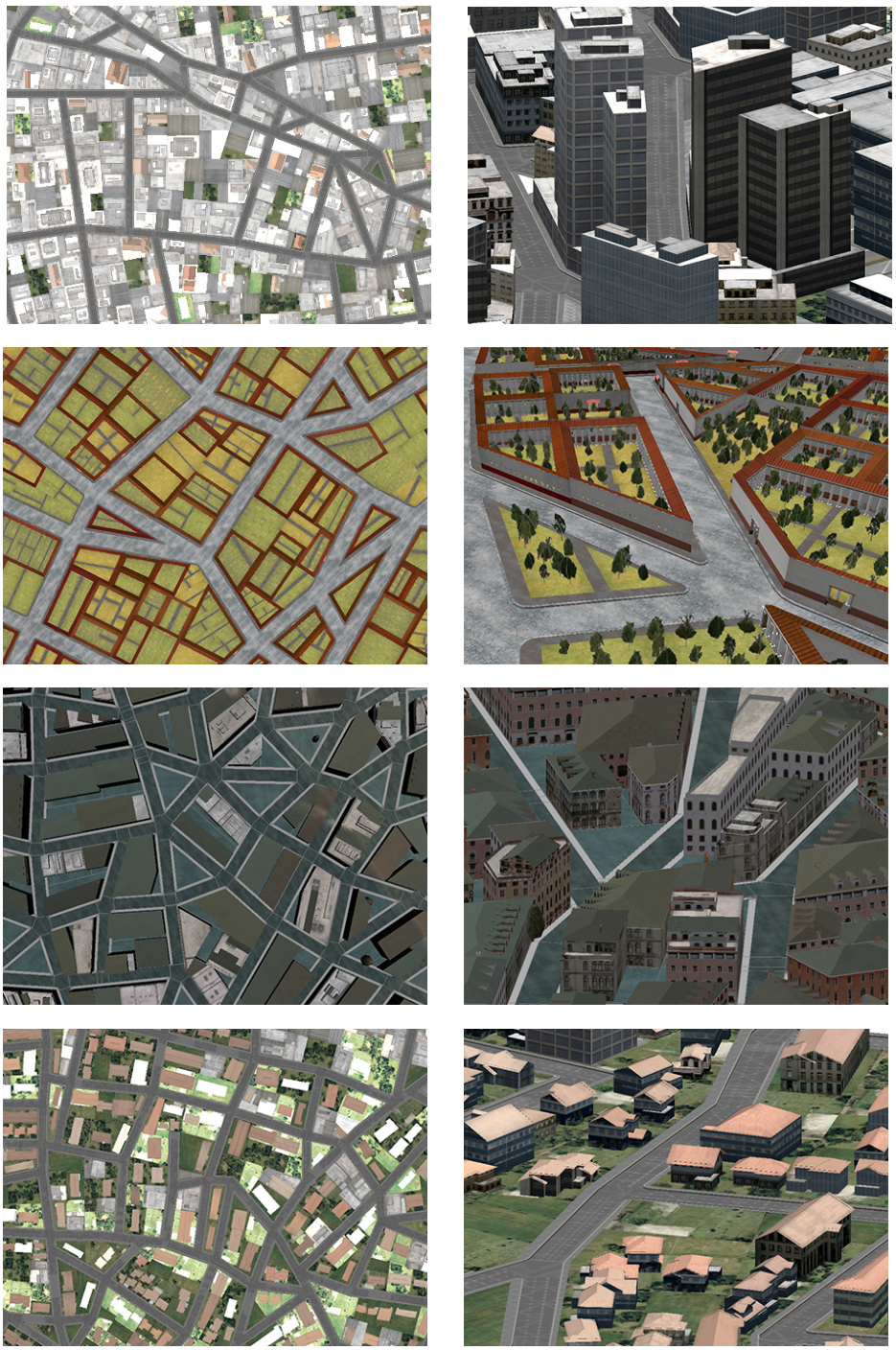

Figure 7.13: **Examples of our Stateless Generation method** Top row: Street network generated from street-network characteristics of Berlin. Detailed view of the generated city; Following rows: Generated cities with alternative CGA grammars - ancient Pompeii [77], Venice and city suburbs

7.13 Future Work

The use of the algorithm presented here is limited to static virtual-world content. The system could be enhanced to handle both static and dynamic content, in which case synchronization among clients will be necessary. Currently, our method does not guarantee an appropriate distribution of street length patterns and angles. The distributions only correspond in mean values. Although the result is already visually attractive, it may be possible to improve the algorithm to simulate the original street patterns more faithfully by reflecting this need more strongly, for example, in the characteristics of the Poisson-disk distribution or in the path selection heuristics.

Thus far, we have not implemented and evaluated the second scenario from section 7.9. We expect that this will be interesting from the viewpoints of scalability and synchronization. Other generating algorithms could be adapted to fulfill Stateless Generation requirements and could be combined with our approach. This would be interesting especially with popular terrain-generating algorithms and engines. Müller et al. [78] have presented an algorithm for synthesizing building façades by example. It would be interesting to combine their approach with ours to automatically generate infinite cities with a structure that matches the maps of existing cities in terms of both building types and street structure.

On-demand real-time generated cities can benefit from techniques such as occlusion detection or level-of-detail of procedurally generated models to increase rendering performance. Previous post-processing tools and techniques must be altered to support models created on-demand. Important progress has been made in a concurrent work by Steinberger et al. [108, 109]. They use GPU for real-time shape-grammar-based generation and rendering of urban landscapes and utilize methods of visibility pruning and adaptive levels of detail to dynamically generate only the geometry needed to render the current view.

Part IV

Closing Part

Chapter 8

Conclusions

This chapter summarizes the results of research presented in this thesis.

8.1 Future of Mobile Graphics

Mobile devices are becoming popular and this field of technology is evolving rapidly. Although new devices have several times more memory and computational power than a few years ago, novel algorithms for saving resources will be still required. This is because a new kinds of wearable mobile devices began to appear. Soon people will be using wearable computers on their wrists, attached to their keys or integrated into their glasses. The size of these devices limits amount of its resources and effective algorithms will be needed. In future we can expect even smaller wearable devices shaped e.g. as a rings or contact lenses. Distributed offloading of computational tasks to cloud servers or a more powerful local mobile device will likely be a necessity.

8.2 Fulfillment of the Goals

The first goal of the thesis was to propose novel methods for semantic-based reduction and procedural generation to allow previously impossible graphical scenarios on mobile devices.

- Our viseme-reduction algorithm uses semantic methods to greatly reduce the amount of memory needed for facial animations and allows creations of fluent talking-head applications that were not possible before.

- The proposed Stateless-Generation algorithm detects which parts of the huge city architecture are visible to the user and avoids consuming resources on currently unnecessary areas.
- The same algorithm also uses procedural generation to create city content on-demand and reduces required data download. Using the algorithm, we can display infinitely large urban landscapes on mobile devices where they were not possible before.

The second goal of the thesis was to improve graphical scenarios in distributed architectures common to mobile devices, considering task offloading and possibly-unreliable wireless connectivity.

- Our framework for creating 3D head applications discusses optimal distribution of tasks between a mobile–client and a server–computer for graphic scenarios with facial animations.
- Our cloud-controlled system for device-to-device content distribution system utilizes systems for cooperative device-to-device video streaming using occasionally-connected opportunistic connections. All participating mobile devices are managed by a central server located in a cloud.
- We proposed Stateless-Generation algorithm that allows the generation and synchronizing of urban architecture on mobile devices even when those devices are not always connected to the server. The algorithm can also be used to allow seamless scaling of resources in a cases of huge procedurally-generated worlds in the cloud.

8.3 Rendering of Facial Models

Using our framework we demonstrated that as mobile clients are becoming more powerful, real-time rendering of a voice-interactive talking head is within their reach. We may expect a boom in voice-interactive 3D mobile applications in fields such as entertainment, commerce, education or virtual assistance. Client-server architecture, rendering and synchronizing 3D and audio components locally and controlling the logic and speech processing remotely, allows applications to be less power-hungry and improves the quality of virtual-character interaction. Our algorithm for viseme reduction is able to reduce size of mesh geometries by more than one third and increases rendering speeds more than twofold. We provided a framework for the easy creation of virtual-character based applications on mobile phones to spark future research and application development in this area. This

framework has been used by a major internet company (Answers.com) to enhance their mobile app with a virtual talking assistant (see. Figure 8.1). We were involved in both the application development and model preparation.

8.4 Collaborative Distributed Computer Graphics

Our cloud-controlled system for device-to-device content distribution showed that cloud logic helps alleviate the saturation of cellular network traffic. Using the cloud, we implemented and experimentally evaluated a novel architecture to disseminate popular video content to subscribed users. Our measurements showed that opportunistic dissemination techniques utilizing a coordinating cloud service can achieve a higher offloading rates than those techniques lacking cloud logic. Using cloud services also allows achieving better effectiveness in environments with large numbers of mobile devices, avoiding the congestion of wireless channels.

In our controlled test, we achieved up to 71% saved bandwidth compared to naive independent downloading approaches that are currently widely used. Our practical experiment showed that having users in the same device-to-device range can save 39% of bandwidth (and a maximum of 53%) by using opportunistic connections when streaming the same live video. Our work makes important progress by demonstrating the feasibility of easily sharing graphical data using the cloud on mobile as well as fixed terminals. Our ultimate goal is to provide an open platform for other researchers to build upon and a tool for the easy deployment of 3D services. This could have huge potential impact in many interactive domains (e-commerce, assistive technologies, real-time interaction, gaming, etc.).

8.5 Virtual Cities on Mobile Devices

Our proposed stateless generation algorithm can generate a possibly infinite street network on-demand. The main advantage of the algorithm is that it can generate only the content that is visible to the client and that the generated content remains consistent in the case of multiple clients. The content maintains consistency even when the network connection of a client becomes unavailable as commonly happens for mobile clients with wireless connections. Our appearance of an infinite city looks more natural than that of the previous works, although it does not perfectly simulate the patterns of the existing street-network example, as the path-selection heuristics do not necessarily result in a perfectly corresponding distribution of segment lengths. The algorithm can be used in real-time both on personal computers and mobile phones.

Figure 8.1: **Animated head models for virtual mobile assistants**: The head models have been created using FaceGen Modeller by Jiri Danihelka.

Bibliography

[1] Navid Abedini, Swetha Sampath, Rajarshi Bhattacharyya, Suman Paul, and Srinivas Shakkottai. Realtime streaming with guaranteed QoS over wireless D2D networks. In *Proceedings of the fourteenth ACM international symposium on Mobile ad hoc networking and computing*, pages 197–206. ACM, 2013.

[2] AcbPocketSoft. acbTaskMan for PocketPC. http://www.acbpocketsoft.com.

[3] Andrea Acquaviva, Emanuele Lattanzi, and Alessandro Bogliolo. *Power-Aware Network Swapping for Wireless Palmtop PCS*, pages 198–213. Springer US, 2004.

[4] S. Agarwal, A. Kumar, A. A. Nanavati, and N. Rajput. The world wide telecom web browser. In *Proceeding of the 17th international conference on World Wide Web*, pages 1121–1122. ACM, 2008.

[5] Sheetal K. Agarwal, Dipanjan Chakraborty, Arun Kumar, Amit Anil Nanavati, and Nitendra Rajput. Hstp: hyperspeech transfer protocol. In *HT '07: Proceedings of the eighteenth conference on Hypertext and hypermedia*, pages 67–76, New York, NY, USA, 2007. ACM.

[6] I. Albrecht, J. Haber, and H. P. Seidel. Speech synchronization for physics-based facial animation. *Proceedings WSCG 2002*, pages 9–16, 2002.

[7] Marc Alexa, Uwe Berner, Michael Hellenschmidt, and Thomas Rieger. An animation system for user interface agents. In *Proceedings of WSCG*, 2001.

[8] D. G. Aliaga, P. A. Rosen, and D. R. Bekins. Style grammars for interactive visualization of architecture. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):786–797, 2007.

[9] D. G. Aliaga, C. A. Vanegas, and B. Beneš. Interactive example-based urban layout synthesis. In *ACM Transactions on Graphics (TOG)*, volume 27, 2008.

[10] Koray Balci. Xface: Mpeg-4 based open source toolkit for 3d facial animation. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 399–402, New York, NY, USA, 2004. ACM.

[11] Koray Balci. *Xface: Open Source Toolkit for Creating 3D Faces of an Embodied Conversational Agent*, pages 263–266. Springer Berlin / Heidelberg, 2005.

[12] Koray Balci. Xfaceed: authoring tool for embodied conversational agents. In *ICMI '05: Proceedings of the 7th international conference on Multimodal interfaces*, pages 208–213, New York, NY, USA, 2005. ACM.

[13] Koray Balci, Elena Not, Massimo Zancanaro, and Fabio Pianesi. Xface open source project and smil-agent scripting language for creating and animating embodied conversational agents. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, pages 1013–1016, New York, NY, USA, 2007. ACM.

[14] Uwe Berner. Optimized face animation with morph-targets. *Journal of WSCG 2004*, 12, 2004.

[15] Fernando Bevilacqua, Cesar Tadeu Pozzer, and Marcos Cordeiro d'Ornellas. Charack: Tool for real-time generation of pseudo-infinite virtual worlds for 3D games. In *Proceedings of the VIII Brazilian Symposium on Games and Digital Entertainment*, pages 111–120. IEEE Computer Society, 2009.

[16] A. W. Black and K. A. Lenzo. Flite: a small fast run-time synthesis engine. In *4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*, pages 20–24, 2001.

[17] D. A. Bowman, S. Coquillart, B. Froehlich, M. Hirose, Y. Kitamura, K. Kiyokawa, and W. Stuerzlinger. 3D User Interfaces: New Directions and Perspectives. *IEEE Computer Graphics and Applications*, 28(6):20–36, 2008.

[18] Scott Burleigh, Adrian Hooke, Leigh Torgerson, Kevin Fall, Vint Cerf, Bob Durst, Keith Scott, and Howard Weiss. Delay-tolerant networking: an approach to interplanetary internet. *Communications Magazine, IEEE*, 41(6):128–136, 2003.

[19] Justine Cassell, Hannes Högni Vilhjálmsson, and Timothy Bickmore. Beat: the behavior expression animation toolkit. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 477–486, New York, 2001. ACM.

[20] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Transactions on Mobile Computing*, 6(6):606–620, 2007.

[21] K. T. Chen, P. Huang, C. Y. Huang, and C. L. Lei. Game traffic analysis: An MMORPG perspective. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 19–24. ACM, 2005.

[22] Soo-Mi Choi, Yong-Guk Kim, Don-Soo Lee, Sung-Oh Lee, and Gwi-Tae Park. Non-photorealistic 3-d facial animation on the PDA based on facial expression recognition. In *Proceedings 4th International Symposium on Smart Graphics, LNCS 3031*, pages 11–20, 2004.

[23] B. G. Chun and P. Maniatis. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th conference on Hot topics in operating systems*, pages 8–11. USENIX Association, 2009.

[24] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Dewall: A fast divide and conquer delaunay triangulation algorithm in E^d. *Computer-Aided Design*, 30(5):333–341, 1998.

[25] Cisco. Visual networking index: Global mobile data traffic forecast update, 2012–2017. White Paper, 2013.

[26] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics*, 22(3):287–294, 2003.

[27] Brian Cullen and Carol O'Sullivan. A caching approach to real-time procedural generation of cities from gis data. *Journal of WSCG*, 19(3):119–126, 2011.

[28] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000.

[29] L. Deng, Y. Wang, K. Wang, A. Acero, H. Hon, J. Droppo, C. Boulis, M. Mahajan, and XD Huang. Speech and language processing for multimodal human-computer interaction. *The Journal of VLSI Signal Processing*, 36(2):161–187, 2004.

[30] Pierre Devevey, Nicolas Lorenzon, and Chaibou Tambary. Measuring wireless energy consumption on PDAs and on laptops. *Universite del la Franche Comte-DISI, Universita di Genova*, 2005.

[31] D. C. Dryer. Getting personal with computers: How to design personalities for agents. *Applied Artifcial Intelligence*, 13(3):273–295, 1999.

[32] Esri. CityEngine – 3D modeling software for urban environments, 2008. http://www.esri.com/software/cityengine.

[33] Frank HP Fitzek and Marcos D Katz. *Cooperation in wireless networks: Principles and applications*. Springer, 2006.

[34] Foreignword.
English-Truespel (USA Accent) Text Conversion Tool.
http://www.foreignword.com/dictionary/truespel/transpel.htm.

[35] G. F. Franklin, J. D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, 2nd edition, 1990.

[36] Gersende Georg, Catherine Pelachaud, and Marc Cavazza. Emotional reading of medical texts using conversational agents. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1285–1288, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[37] Kevin R Glass, Chantelle Morkel, and Shaun D Bangay. Duplicating road patterns in south african informal settlements using procedural techniques. In *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 161–169. ACM, 2006.

[38] S. Greuter, J. Parker, N. Stewart, and G. Leach. Real-time procedural generation of pseudo infinite cities. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 87–95. ACM, 2003.

[39] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Undiscovered worlds – Towards a framework for real-time procedural world generation. In *Fifth International Digital Arts and Culture Conference, Melbourne, Australia*, 2003.

[40] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

[41] Bo Han, Pan Hui, VS Kumar, Madhav V Marathe, Guanhong Pei, and Aravind Srinivasan. Cellular traffic offloading through opportunistic communications: a case

study. In *Proceedings of the 5th ACM workshop on Challenged networks*, pages 31–38. ACM, 2010.

[42] Sha Hua, Yang Guo, Yong Liu, Hang Liu, and Shivendra S Panwar. Scalable video multicast in hybrid 3g/ad-hoc networks. *IEEE Transactions on Multimedia*, 13(2):402–413, 2011.

[43] D. Huggins-Daines, M. Kumar, A. Chan, AW Black, M. Ravishankar, and AI Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP Proceedings*, volume 1, 2006.

[44] Suk Yu Hui and Kai Hau Yeung. Challenges in the migration to 4g mobile systems. *Communications Magazine, IEEE*, 41(12):54–59, 2003.

[45] A. Iosup, A. Lăscăteu, and N. Ţăpuş. Cameo: enabling social networks for massively multiplayer online games through continuous analytics and cloud computing. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, page 7. IEEE Press, 2010.

[46] ISO/IEC 14496-1. *Information technology – Coding of audio-visual objects – Part 1: Systems*. ISO, Geneva, Switzerland, 1999.

[47] ISO/IEC 14496-2. *Information technology – Coding of audio-visual objects – Part 2: Visual*. ISO, Geneva, Switzerland, 1999.

[48] Eric Jung, Yichuan Wang, Iuri Prilepov, Frank Maker, Xin Liu, and Venkatesh Akella. User-profile-driven collaborative bandwidth sharing on mobile phones. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 2. ACM, 2010.

[49] M.W. Kadous and C. Sammut. Mobile conversational characters. *HF2002: Virtual Conversational Characters: Applications, Methods, and Research Challenge, Melbourne, Australia*, 2002.

[50] Gunnar Karlsson, Vincent Lenders, and Martin May. Delay-tolerant broadcasting. *IEEE Transactions on Broadcasting*, 53(1):369–381, 2007.

[51] Lorenzo Keller, Anh Le, Blerim Cici, Hulya Seferoglu, Christina Fragouli, and Athina Markopoulou. Microcast: cooperative video streaming on smartphones. In

Proceedings of the 10th international conference on Mobile systems, applications, and services, pages 57–70. ACM, 2012.

[52] C. Keskin, K. Balci, O. Aran, B. Sankur, and L. Akarun. A multimodal 3D healthcare communication system. In *3DTV Conference*, pages 1–4, 2007.

[53] Kishonti Informatics. GL Benchmark. http://glbenchmark.com.

[54] Donald Ervin Knuth. *The art of computer programming 4th edition, volume 2, section 3.2*. Addison-Wesley, 2006.

[55] Jonathan G. Koommey, Stephen Berard, Marla Sanchez, and Henry Wong. Assessing trends in the electrical efficiency of computation over time. *IEEE Annals of the History of Computing*, 2009.

[56] Kronous Groups. OpenGL ES - The Standard for Embedded Accelerated 3D Graphics. http://www.khronos.org/opengles/.

[57] Mike Krus, Patrick Bourdot, Françoise Guisnel, and Gullaume Thibault. Levels of detail & polygonal simplification. *Crossroads*, 3(4):13–19, 1997.

[58] Sumedha Kshirsagar, Nadia Magnenat-Thalmann, Anthony Guye-Vuillème, Daniel Thalmann, Kaveh Kamyab, and Ebrahim Mamdani. Avatar markup language. In *EGVE '02: Proceedings of the workshop on Virtual environments*, pages 169–177, Aire-la-Ville, Switzerland, 2002. Eurographics Association.

[59] Arun Kumar, Nitendra Rajput, Dipanjan Chakraborty, Sheetal K. Agarwal, and Amit A. Nanavati. Wwtw: the world wide telecom web. In *NSDR '07: Proceedings of the workshop on Networked systems for developing regions*, pages 1–6, New York, NY, USA, 2007. ACM.

[60] Ladislav Kunc and Jan Kleindienst. *ECAF: Authoring Language for Embodied Conversational Agents*, pages 206–213. Springer, 2007.

[61] Ladislav Kunc, Pavel Slavik, and Jan Kleindienst. Talking head as life blog. In *Text, Speech and Dialogue*, Lecture Notes in Computer Science, pages 365–372, 2008.

[62] Geo Leach. Improving worst-case optimal delaunay triangulation algorithms. In *4th Canadian Conference on Computational Geometry*, pages 340–346. Citeseer, 1992.

[63] T. Lechner, B. Watson, U. Wilensky, and M. Felsen. Procedural modeling of land use in cities. In *Midgraph Conference, Washington University, St. Louis, MO*. Citeseer, 2003.

[64] S. Lefebvre and H. Hoppe. Parallel controllable texture synthesis. *ACM Transactions on Graphics (TOG)*, 24(3):777–786, 2005.

[65] Vincent Lenders, Gunnar Karlsson, and Martin May. Wireless ad hoc podcasting. In *4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks SECON'07*, pages 273–283. IEEE, 2007.

[66] Zuo Li, LI Jin-tao, and Wang Zhao-qi. Anatomical human musculature modeling for real-time deformation. *Journal of WSCG 2003*, 11, 2003.

[67] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (ToG)*, 20(3):127–150, 2001.

[68] Wallace B McClure, Nathan Blevins, John J Croft IV, et al. *Cross Platform Android and iOS Mobile Development*. Wrox, 2012.

[69] S. Melax. A simple, fast, and effective polygon reduction algorithm. *Game Developer*, 11:44–49, 1998.

[70] P. Merrell. Example-based model synthesis. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 105–112. ACM, 2007.

[71] P. Merrell and D. Manocha. Continuous model synthesis. In *ACM Transactions on Graphics (TOG)*, 2008.

[72] Microsoft. Windows Azure. http://www.microsoft.com/windowsazure.

[73] Microsoft. Windows Phone 7. http://www.windowsphone7.com.

[74] Microsoft. Silverlight, 2007. http://www.silverlight.net/.

[75] Bren Mochocki, Kanishka Lahiri, and Srihari Cadambi. Power analysis of mobile 3d graphics. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 502–507, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[76] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. *Procedural modeling of buildings*, volume 25. ACM, 2006.

[77] Pascal Müller, Tijl Vereenooghe, Andreas Ulmer, and Luc Van Gool. Automatic reconstruction of Roman housing architecture. *Recording, modeling and visualization of cultural heritage*, pages 287–298, 2005.

[78] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. *ACM Transactions on Graphics*, 26(3), 2007.

[79] M. T. Najaran and C. Krasic. Scaling online games with adaptive interest management in the cloud. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, page 9. IEEE Press, 2010.

[80] Clifford Nass, Youngme Moon, B. J. Fogg, Byron Reeves, and Chris Dryer. Can computer personalities be human personalities? In *CHI '95: Conference companion on Human factors in computing systems*, pages 228–229, New York, NY, USA, 1995. ACM.

[81] Radoslaw Niewiadomski, Elisabetta Bevacqua, Maurizio Mancini, and Catherine Pelachaud. Greta: an interactive expressive ECA system. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1399–1400, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[82] Amalia Ortiz, Maria del Puy Carretero, David Oyarzun, Jose Yanguas, Cristina Buiza, M. Gonzalez, and Igone Etxeberria. *Elderly Users in Ambient Intelligence: Does an Avatar Improve the Interaction?*, pages 99–114. Springer Berlin / Heidelberg, 2007.

[83] Igor S. Pandzic. Facial animation framework for the web and mobile platforms. In *Web3D '02: Proceedings of the seventh international conference on 3D Web technology*, pages 27–34, New York, USA, 2002. ACM.

[84] Igor S. Pandzic and Robert Forchheimer. *MPEG-4 Facial Animation: The Standard, Implementation and Applications*, pages 15–61. Wiley, 2002.

[85] I.S. Pandzic, J. Ahlberg, M. Wzorek, P. Rudol, and M. Mosmondor. Faces everywhere: towards ubiquitous production and delivery of face animation. In *Proceedings of the 2nd International Conference on Mobile and Ubiquitous Multimedia, Norrkoping, Sweden*, 2003.

[86] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

[87] W. Pasman and F. W. Jansen. Scheduling level of detail with guaranteed quality and cost. In *Web3D '02: Proceedings of the seventh international conference on 3D Web technology*, pages 43–51, New York, NY, USA, 2002. ACM.

[88] Catherine Pelachaud. Multimodal expressive embodied conversational agents. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 683–689, New York, NY, USA, 2005. ACM.

[89] Kostas Pentikousis. In search of energy-efficient mobile networking. *IEEE Communications Magazine*, 48(1):95–103, 2010.

[90] Charles Petzold. *Microsoft XNA Framework Edition: Programming Windows Phone 7*. Microsoft press, 2010.

[91] P. Poller and J. Muller. Distributed audio-visual speech synchronization. In *Seventh International Conference on Spoken Language Processing*, 2002.

[92] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer, 1991.

[93] P. Prusinkiewicz, A. Lindenmayer, J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer. *The algorithmic beauty of plants*. Springer New York, 1990.

[94] Kari Pulli, Jani Vaarala, Ville Miettinen, Robert Simpson, Tomi Aarnio, and Mark Callow. The mobile 3d ecosystem. In *ACM SIGGRAPH courses*, page 1, New York, NY, USA, 2007. ACM.

[95] Lingyun Qiu and Izak Benbasat. An investigation into the effects of text-to-speech voice and 3d avatars on the perception of presence and flow of live help in electronic commerce. *ACM Trans. Comput.-Hum. Interact.*, 12(4):329–355, 2005.

[96] Qt Software. Qt Application Framework. http://trolltech.com/products.

[97] Marwan Ramadan, Layla El Zein, and Zaher Dawy. Implementation and evaluation of cooperative video streaming for mobile devices. In *IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–5. IEEE, 2008.

[98] I. V. Ramakrishnan, Amanda Stent, and Guizhen Yang. Hearsay: enabling audio browsing on hypertext content. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 80–89, New York, NY, USA, 2004. ACM.

[99] M. Reddy. SCROOGE: Perceptually-driven polygon reduction. In *Computer Graphics Forum*, volume 15, pages 191–203. John Wiley & Sons, 2003.

[100] Matthew Seligman, Kevin Fall, and Padma Mundur. Alternative custodians for congestion control in delay tolerant networks. In *Proceedings of the SIGCOMM workshop on Challenged networks*, pages 229–236. ACM, 2006.

[101] Sujan Shrestha. Mobile web browsing: usability study. In *Mobility '07: Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, pages 187–194, New York, NY, USA, 2007. ACM.

[102] Eftychios Sifakis, Andrew Selle, Avram Robinson-Mosher, and Ronald Fedkiw. Simulating speech with a physics-based facial muscle model. In *SCA '06: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 261–270, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[103] Olli Silven and Kari Jyrkka. Observations on power-efficiency trends in mobile communication devices. *EURASIP Journal on Embedded Systems*, 2007.

[104] Singular Inversion. FaceGen. www.facegen.com.

[105] A Prasad Sistla, Ouri Wolfson, and Bo Xu. Opportunistic data dissemination in mobile peer-to-peer networks. In *Advances in Spatial and Temporal Databases*, pages 346–363. Springer, 2005.

[106] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259. ACM, 2005.

[107] O. Šťava, B. Beneš, R. Měch, D. G. Aliaga, and P. Krištof. Inverse procedural modeling by automatic generation of L-systems. In *Computer Graphics Forum*, volume 29, pages 665–674. Wiley Online Library, 2010.

[108] Markus Steinberger, Michael Kenzel, Bernhard Kainz, Joerg Mueller, Peter Wonka, and Dieter Schmalstieg. Parallel generation of architecture on the GPU. In *Computer Graphics Forum*, volume 33, 2014.

[109] Markus Steinberger, Michael Kenzel, Bernhard Kainz, Peter Wonka, and Dieter Schmalstieg. On-the-fly generation and rendering of infinite cities on the GPU. In *Computer Graphics Forum*, volume 33, 2014.

[110] Jing Sun, Xiaobo Yu, George Baciu, and Mark Green. Template-based generation of road networks for virtual city modeling. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 33–40. ACM, 2002.

[111] Zan Sun, Amanda Stent, and I. V. Ramakrishnan. Dialog generation for voice browsing. In *W4A '06: Proceedings of the international cross-disciplinary workshop on Web accessibility (W4A)*, pages 49–56, New York, NY, USA, 2006. ACM.

[112] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[113] C. A. Vanegas, D. G. Aliaga, B. Beneš, and P. A. Waddell. Interactive design of urban spaces using geometrical and behavioral modeling. In *ACM SIGGRAPH Asia papers*, pages 1–10. Citeseer, 2009.

[114] C. A. Vanegas, D. G. Aliaga, P. Wonka, P. Müller, P. Waddell, and B. Watson. Modelling the appearance and behaviour of urban spaces. In *Computer Graphics Forum*, volume 29, pages 25–42. Wiley Online Library, 2010.

[115] Carlos A Vanegas, Tom Kelly, Basil Weber, Jan Halatsch, Daniel G Aliaga, and Pascal Müller. Procedural generation of parcels in urban modeling. In *Computer Graphics Forum*, volume 31, pages 681–690. Wiley Online Library, 2012.

[116] Daniel Wagner, Mark Billinghurst, and Dieter Schmalstieg. How real should virtual characters be? In *ACE '06: Proceedings of the ACM SIGCHI international conference on Advances in computer entertainment technology*, page 57, New York, NY, USA, 2006. ACM.

[117] J. Waldo. Scaling in games and virtual worlds. *Communications of the ACM*, 51(8):38–44, 2008.

[118] Alice Wang, Michael Emmi, and Petros Faloutsos. Assembling an expressive facial animation system. In *Sandbox '07: Proceedings of the ACM SIGGRAPH symposium on Video games*, pages 21–26, New York, NY, USA, 2007. ACM.

[119] K. Waters and T.M. Levergood. DECface: An automatic lip-synchronization algorithm for synthetic faces. *Digital Equipment Corp, Cambridge Research Laboratory, Technical Report Series 93*, 4, 1993.

[120] B. Weber, P. Muller, P. Wonka, and M. Gross. Interactive geometric simulation of 4D cities. In *Computer Graphics Forum*, volume 28, pages 481–492. Blackwell Publishing, 2009.

[121] John Whitbeck, Marcelo Amorim, Yoann Lopez, Jeremie Leguay, and Vania Conan. Relieving the wireless infrastructure: When opportunistic networks meet guaranteed delays. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–10. IEEE, 2011.

[122] A. Wigley, M. Sutton, S. Wheelwright, R. Burbidge, and R. Mcloud. *Microsoft. Net Compact Framework: Core Reference*. Microsoft Press Redmond, WA, USA, 2002.

[123] Hanno Wirtz, Jan Rüth, Torsten Zimmermann, and Klaus Wehrle. Interest-based cloud-facilitated opportunistic networking. In *Proceedings of the 8th ACM MobiCom workshop on challenged networks*, pages 63–68. ACM, 2013.

[124] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. *Instant architecture*, volume 22. ACM, 2003.

[125] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on computer graphics and interactive techniques*, pages 291–294. ACM, 1996.

[126] Min Yin and Shumin Zhai. The benefits of augmenting telephone voice menu navigation with visual browsing and search. In *CHI '06: Proceedings of the SIGCHI conference on human factors in computing systems*, pages 319–328, New York, NY, USA, 2006. ACM.

Author's publications related to the thesis

Publications in journals

[A.1] Jiri Danihelka, Lukas Kencl, and Jiri Zara
Stateless generation of distributed virtual worlds
In *Computers & Graphics Journal*, volume 44, pages 33-44, Elsevier, 2014.
12 pages
WoS citations: 0 Google Scholar citations: 0
Journal impact factor: 1.029
(Ratio: 75%, 15%, 10%) [Indexed in Web of Science]

[A.2] Jiri Danihelka, Roman Hak, Lukas Kencl, and Jiri Zara
3D talking-head interface to voice-interactive services on mobile phones
In *Mobile HCI SiMPE 2010 Papers Proceedings*, 2010
8 pages

The paper has been selected as the best paper and published in a journal:
International Journal of Mobile Human Computer Interaction (IJMHCI)
pages 50-64, volume 3, IGI global, 2011
14 pages
Journal impact factor: 0.15

Consequently the paper has been published in a book:
Developments in Technologies for Human-Centric Mobile Computing and Applications
pages 130-144, ISBN13 9781466620681, ISBN10 1466620684, IGI global, 2012
14 pages
WoS citations: 3 Google Scholar citations: 9
(Ratio: 40%, 30%, 20%, 10%)

Publications indexed in Web of Science and SCOPUS

[A.3] Jiri Danihelka, and Lukas Kencl

Collaborative 3D environments over Windows Azure

Proceedings of the IEEE 7th International Symposium on Service Oriented System Engineering (SOSE), 2013.

6 pages

WoS citations: 2 Google Scholar citations: 3

(Ratio: 60%, 40%) [Indexed in Web of Science, Scopus]

[A.4] Jiri Danihelka, Domenico Giustiniano, and Theus Hossmann

HyCloud: A system for device-to-device content distribution controlled by the cloud

In *Proceedings of The 15th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM 2014)* , 16-19 June 2014, Sydney, Australia.

5 pages

WoS citations: 0 Google Scholar citations: 0

(Ratio: 70%, 15%, 15%) [Indexed in Web of Science]

[A.5] Jiri Danihelka, Lukas Kencl, and Jiri Zara

Reduction of animated models for embedded devices

In *WSCG 2010 Communication Papers Proceedings*, 2010.

5 pages

WoS citations: 3 Google Scholar citations: 8

(Ratio: 80%, 10%, 10%) [Indexed in Web of Science]

Other publications

[A.6] Jiri Danihelka, Domenico Giustiniano, and Bernhard Plattner
On a Cloud-Controlled Architecture for Device-to-Device Content Distribution
In *Proceedings of the 10th ACM workshop on Challenged networks (ACM CHANTS 2015)* , 11th September 2015, Paris, France.
6 pages
(Ratio: 50%, 40%, 10%) [To appear]

[A.7] David Sedlacek, Jiri Danihelka, Zdenek Travnicek, Michal Lukac, Roman Berka, Jiri Zara
Virtual Cities in Time And Space (ViCiTiS)
CTU in Prague: Department of Computer Graphics and Interaction
CS-TR-DCGI-2012-4, ISSN 1805-6180, 2012
53 pages, technical report
WoS citations: 1 Google Scholar citations: 0
(Ratio: 25%, 25%, 15%, 15%, 10%, 10%)

[A.8] Jiri Danihelka, and Jiri Zara
Procedural generation of infinite cities
In *Proceedings of the Eurographics Conference 2011*, pages 31–33, The Eurographics Association.
3 pages + poster
(Ratio: 80%, 20%)

[A.9] Jiri Danihelka, and Lukas Kencl
Interactive 3D environments over Windows Azure
In *Proceedings of the Cloud Futures Workshop 2012*, Berkeley, California, USA, 2012.
5 pages
(Ratio: 60%, 40%)

[A.10] Jiri Danihelka, Roman Hak, Lukas Kencl, Jiri Zara

Demo: Interacting with a 3D talking-head on a mobile phone

In *Mobile HCI SiMPE 2010 Demo Papers Proceedings*, 2010.

1 page

(Ratio: 40%, 30%, 20%, 10%)

[A.11] Jiri Danihelka, Lukas Kencl, Roman Hak

Talking with an avatar on a mobile client

In *International Conference on Advances in Computer Entertainment Technology, Salon de ACE*. ACM, 2009.

poster

(Ratio: 50%, 25%, 25%)

[A.12] Jiri Danihelka, Lukas Kencl, Roman Hak

Client-server talking head on mobile

Science beyond Fiction: The European Future Technologies Conference (FET09)

Prague 2009

poster

(Ratio: 50%, 25%, 25%)

[A.13] Jiri Danihelka, Roman Hak, Lukas Kencl, Jiri Zara

Talking avatar on PDA device

Poster 2010 - 14th International Student Conference on Electrical Engineering

CTU Prague 2010

2 pages and poster

(Ratio: 40%, 30%, 20%, 10%)

Author's publications not related to the thesis

Publications indexed in Web of Science and SCOPUS

[A.14] Katerina Dufkova, Jiri Danihelka, Michal Ficek, Ivan Gregor, and Jan Kouba
Can active tracking of inroamer location optimise a live GSM network?
In *Proceedings of the 2007 ACM CoNEXT conference*, ISBN: 978-1-59593-770-4, ACM, 2007.
2 pages + poster
WoS citations: 1 Google Scholar citations: 3
(Ratio: 20%, 20%, 20%, 20%, 20%) [Indexed in Scopus]

[A.15] Katerina Dufkova, Michal Ficek, Lukas Kencl, Jan Novak, Jan Kouba, Ivan Gregor, and Jiri Danihelka
Active GSM Cell-ID tracking: Where did you disappear?
In *Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments*, pages 7–12. ACM, 2008.
6 pages
WoS citations: 5 Google Scholar citations: 21
(Ratio: 20%, 20%, 20%, 10%, 10%, 10%, 10%) [Indexed in Scopus]

Organized conferences

[A.16] **Microsoft Fest 2012, Czech Technical University in Prague**
Chairman: Jiri Danihelka
Keynote to mobile development track: **XAML language**
Second talk: **Advanced UI design with Expression Blend**
http://www.ms-fest.cz/2012/

Citations of publications

Self-citations are included.

Web of Science citations of [A.2]

[C.1] Jiri Danihelka and Lukas Kencl. Collaborative 3d environments over Windows Azure. In *IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, pages 472–477. IEEE, 2013.

[C.2] Roman Hak, Jakub Dolezal, and Tomas Zeman. Manitou: A multimodal interaction platform. In *5th Joint IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 60–63. IEEE, 2012.

[C.3] Cristian Negrescu, Amelia Ciobanu, and Marija D. Ilie. Specific acoustic unit processing in concatenative romanian speech synthesis used for talking agents. In *7th Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, pages 1–8. IEEE, 2013.

Google Scholar citations of [A.2]

[C.4] Jiri Danihelka and Lukas Kencl. Interactive 3d environments over Windows Azure. In *Proceedings of the Cloud Futures Workshop*, 2012.

[C.5] Jiri Danihelka and Lukas Kencl. Collaborative 3d environments over Windows Azure. In *IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, pages 472–477. IEEE, 2013.

[C.6] Roman Hak, Jakub Dolezal, and Tomas Zeman. Manitou: A multimodal interaction platform. In *5th Joint IFIP on Wireless and Mobile Networking Conference (WMNC)*, pages 60–63. IEEE, 2012.

[C.7] Mohd Najib Hamdan and Ahmad Zamzuri Mohamad Ali. User satisfaction of non-realistic three-dimensional talking-head animation courseware (3d-nr).

[C.8] Mohd Najib Hamdan and Ahmad Zamzuri Mohamad Ali. Developing and evaluating of non-realistic three-dimensional (3d-nr) and two-dimensional (2d) talking-head animation courseware. *Malaysian Online Journal of Educational Technology*, 2015.

[C.9] Marija D Ilie, Amelia Ciobanu, Cristian Negrescu, and Dumitru Stanomir. Towards expressive romanian speaking 3d avatars for multimedia interfaces. In *20th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 47–50. IEEE, 2013.

[C.10] Huijie Lin, Jia Jia, Xiangjin Wu, and Lianhong Cai. Talkingandroid: An interactive, multimodal and real-time talking avatar application on mobile phones. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 1–4. IEEE, 2013.

[C.11] Cristian Negrescu, Amelia Ciobanu, and Marija D Ilie. Specific acoustic unit processing in concatenative romanian speech synthesis used for talking agents. In *7th Conference Speech Technology and Human-Computer Dialogue (SpeD)*, pages 1–8. IEEE, 2013.

[C.12] Marcos Santos Pérez. *Análisis y Optimización de Agentes Conversacionales 3D para Sistemas Empotrados*. PhD thesis, Universidad de Málaga, 2014.

Web of Science citations of [A.3]

[C.13] Jiri Danihelka, Lukas Kencl, and Jiri Zara. Stateless generation of distributed virtual worlds. *Computers & Graphics*, 44:33–44, 2014.

[C.14] Ondrej Tomanek and Lukas Kencl. Claudit: Planetary-scale cloud latency auditing platform. In *IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pages 138–146. IEEE, 2013.

Google Scholar citations of [A.3]

[C.15] José López Cívico. Measuring latency in Windows Azure using a collaborative 3d environment. 2013.

[C.16] Jiri Danihelka, Lukas Kencl, and Jiri Zara. Stateless generation of distributed virtual worlds. *Computers & Graphics*, 44:33–44, 2014.

[C.17] Ondrej Tomanek and Lukas Kencl. Claudit: Planetary-scale cloud latency auditing platform. In *IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pages 138–146. IEEE, 2013.

Web of Science citations of [A.5]

[C.18] Jiri Danihelka and Lukas Kencl. Collaborative 3d environments over Windows Azure. In *IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, pages 472–477. IEEE, 2013.

[C.19] Engin Mendi. A 3d face animation system for mobile devices. *Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology*, 26(1):11–18, 2014.

[C.20] Engin Mendi and Coskun Bayrak. Text-to-audiovisual speech synthesizer for children with learning disabilities. *Telemedicine and e-Health*, 19(1):31–35, 2013.

Google Scholar citations of [A.5]

[C.21] Jiri Danihelka, Roman Hak, Lukas Kencl, and Jiri Zara. 3d talking-head interface to voice-interactive services on mobile phones. *Developments in Technologies for Human-Centric Mobile Computing and Applications*, page 130, 2012.

[C.22] Jiri Danihelka and Lukas Kencl. Interactive 3d environments over Windows Azure. In *Proceedings of the Cloud Futures Workshop*, 2012.

[C.23] Jiri Danihelka and Lukas Kencl. Collaborative 3d environments over Windows Azure. In *7th International Symposium on Service Oriented System Engineering (SOSE), 2013* , pages 472–477. IEEE, 2013.

[C.24] Bc Jakub Doležal. Openspeechplatform: Návrh a implementace komplexní hlasové aplikace.

[C.25] Engin Mendi. A 3d face animation system for mobile devices. *Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology*, 26(1):11–18, 2014.

[C.26] Engin Mendi and Coskun Bayrak. Facial animation framework for web and mobile platforms. In *13th International Conference on e-Health Networking Applications and Services (Healthcom)*, pages 52–55. IEEE, 2011.

[C.27] Engin Mendi and Coskun Bayrak. Text-to-audiovisual speech synthesizer for children with learning disabilities. *Telemedicine and e-Health*, 19(1):31–35, 2013.

[C.28] Matthew David Ramage. Disproving visemes as the basic visual unit of speech. 2013.

Web of Science citations of [A.7]

[C.29] Jiri Danihelka, Lukas Kencl, and Jiri Zara. Stateless generation of distributed virtual worlds. *Computers & Graphics*, 44:33–44, 2014.

Web of Science citations of [A.14]

[C.30] Michal Ficek, Tomáš Pop, and Lukáš Kencl. Active tracking in mobile networks: An in-depth view. *Computer Networks*, 57(9):1936–1954, 2013.

Google Scholar citations of [A.14]

[C.31] Michal Ficek. Tracking users in mobile networks: Data acquisition methods and their limits. 2013.

[C.32] Michal Ficek, Tomáš Pop, and Lukáš Kencl. Active tracking in mobile networks: An in-depth view. *Computer Networks*, 57(9):1936–1954, 2013.

[C.33] Jakub Novák and Jana Temelová. Každodenní život a prostorová mobilita mladých Pražanů: pilotní studie využití lokalizačních dat mobilních telefonů. *Sociologický časopis/Czech Sociological Review*, (05):911–938, 2012.

Web of Science citations of [A.15]

[C.34] Kateřina Dufková, Jean-Yves Le Boudec, Lukáš Kencl, and Milan Bjelica. Predicting user-cell association in cellular networks from tracked data. In *Mobile Entity Localization and Tracking in GPS-less Environnments*, pages 19–33. Springer, 2009.

[C.35] Jakub Novak and Jana Temelova. Everyday life and spatial mobility of young people in prague: a pilot study using mobile phone location data. *Sociologicky Casopis-Czech Sociological Review*, 48(5):911–938, 2012.

[C.36] Guang Yang. Discovering significant places from mobile phones–a mass market solution. In *Mobile Entity Localization and Tracking in GPS-less Environnments*, pages 34–49. Springer, 2009.

Google Scholar citations of [A.15]

[C.37] Carlos Azevedo, Gorete Dinis, and Zélia Breda. Understanding visitors' spatio-temporal distribution through data collection using information and communication technologies.

[C.38] Kateřina Dufková, Milan Bjelica, Byongkwon Moon, Lukáš Kencl, and J.-Y. Le Boudec. Energy savings for cellular network with evaluation of impact on data traffic performance. In *Wireless Conference (EW), 2010 European*, pages 916–923. IEEE, 2010.

[C.39] Kateřina Dufková, Jean-Yves Le Boudec, Lukáš Kencl, and Milan Bjelica. Predicting user-cell association in cellular networks from tracked data. In *Mobile Entity Localization and Tracking in GPS-less Environnments*, pages 19–33. Springer, 2009.

[C.40] Michal Ficek. Tracking users in mobile networks: Data acquisition methods and their limits. 2013.

[C.41] Michal Ficek and Lukáš Kencl. Improving roamer retention by exposing weak locations in gsm networks. In *Proceedings of the 5th international student workshop on Emerging networking experiments and technologies*, pages 17–18. ACM, 2009.

[C.42] Michal Ficek and Lukas Kencl. Inter-call mobility model: a spatio-temporal refinement of call data records using a gaussian mixture model. In *INFOCOM proceedings*, pages 469–477. IEEE, 2012.

[C.43] Michal Ficek, Tomáš Pop, and Lukáš Kencl. Active tracking in mobile networks: An in-depth view. *Computer Networks*, 57(9):1936–1954, 2013.

[C.44] Michal Ficek, Tomáš Pop, Petr Vláčil, Kateřina Dufková, Lukáš Kencl, and Martin Tomek. Performance study of active tracking in a cellular network using a modular signaling platform. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 239–254. ACM, 2010.

[C.45] Matthew Kwan. *Visualization and analysis of mobile phone location data*. PhD thesis, RMIT University, 2012.

[C.46] Kari Laasonen et al. Mining cell transition data. 2009.

[C.47] Teddy Mantoro, Abi Dzar Jaafar, Mohd Fadhli Md Aris, Media Ayu, et al. Hajjlocator: A hajj pilgrimage tracking framework in crowded ubiquitous environment. In *International Conference on Multimedia Computing and Systems (ICMCS)*, pages 1–6. IEEE, 2011.

[C.48] Eduardo Baena Martinez, Michal Ficek, and Lukas Kencl. Mobility data anonymization by obfuscating the cellular network topology graph. In *International Conference on Communications (ICC)*, pages 2032–2036. IEEE, 2013.

[C.49] Abdel Meniem, H. Mohamed, Ahmed M. Hamad, and Eman Shaaban. Relative RSS-based GSM localization technique. In *Electro/Information Technology (EIT), 2013 IEEE International Conference*, pages 1–6. IEEE, 2013.

[C.50] Abdel Meniem, H. Mohamed , Ahmed M. Hamad, and Eman Shaaban. GSM-based positioning technique using relative received signal strength. *International Journal of Handheld Computing Research (IJHCR)*, 4(4):38–51, 2013.

[C.51] Jakub Novák and Jana Temelová. Každodenní život a prostorová mobilita mladých Pražanů: pilotní studie využití lokalizačních dat mobilních telefonů. *Sociologický časopis/Czech Sociological Review*, (05):911–938, 2012.

[C.52] Biplav Srivastava, Tran Viet Huan, Wei Xiong Shang, Ullas Nambiar, Vivek Tyagi, and Shivkumar Kalyanaraman. Towards a sustainable services ecosystem for traffic management. In *Annual SRII Global Conference (SRII)*, pages 392–400. IEEE, 2011.

[C.53] RRSS Strength. Practical positioning and traffic estimation techniques using relative received signal.

[C.54] Keen Sung, Brian Neil Levine, and Marc Liberatore. Location privacy without carrier cooperation. *Proc. IEEE*.

[C.55] Shang Weixiong, Zhu Yanfeng, Zhou Jin, and Ying Chun. Collecting and analyzing mobility data from mobile network. In *IC-BNMT'09. 2nd IEEE International Conference on Broadband Network & Multimedia Technology*, pages 810–815. IEEE, 2009.

[C.56] Kuldeep Yadav and Vinayak Naik. Geo-localization and location-aware opportunistic communication for mobile phones. 2014.

[C.57] Guang Yang. Discovering significant places from mobile phones–a mass market solution. In *Mobile Entity Localization and Tracking in GPS-less Environnments*, pages 34–49. Springer, 2009.

List of Supervised Students

These students were supervised by Jiri Danihelka during his PhD study.

1. Michal Holanec
   Comparison of programs for editing 2D graphics
   bachelor thesis

2. Jan Kvasnička
   Sign language visualization on PDA
   bachelor thesis

3. Jan Pavlovský
   Virtual worlds on PDAs
   bachelor thesis

4. Martin Svatek
   Interactive model of RDC lab
   bachelor thesis

5. Jan Zíka
   Model of the GSM network using VRML
   bachelor thesis

6. Petr Švestka
   Usage of interface agents
   diploma thesis

7. Marek Loucký
   Virtual worlds gallery
   bachelor thesis

8. Matous Bednář
   Demo applications for flystick
   bachelor thesis

9. Aleš Havlíček
   The use of technology VRML for sale of goods and services offer
   bachelor thesis

10. Rymzhan Bayekeyeva
    Graphic libraries for Windows Mobile
    bachelor thesis

11. Jaroslav Minařík
    Connection between Cave and CityEngine
    bachelor thesis

12. Radek Sedláček
    Virtual customer care center
    bachelor thesis

13. Jakub Vampola
    Grammar for the generating current Prague architecture
    bachelor thesis

14. Martin Šembera
    The modeler's guide in the CGA language
    bachelor thesis

All bachelor and diploma theses are available online on the Czech Technical University web page: `https://dip.felk.cvut.cz/`

List of Figures

List of Tables