Technical Report Series of DCGI, Volume 2, Year 2012

Department of Computer Graphics and Interaction
Czech Technical University in Prague, CZ
Faculty of Electrical Engineering
Website: http://dcgi.fel.cvut.cz

Virtual Cities in Time And Space (ViCiTiS)

David Sedláček, Jiří Danihelka, Zdeněk Trávníček, Michal Lukáč, Roman Berka, Jiří Žára

CS–TR–DCGI–2012–4 December 2012

Authors' Addresses

David Sedláček
E-mail: david.sedlacek AT fel DOT cvut DOT cz
Web: `http://dcgi.felk.cvut.cz/people/sedlad1`

Jiří Danihelka
E-mail: danihjir AT fel DOT cvut DOT cz
Web: `http://webdev.felk.cvut.cz/~danihjir/`

Zdeněk Trávníček

Michal Lukáč

Roman Berka

Jiří Žára

Department of Computer Graphics and Interaction
Czech Technical University in Prague, FEE
Czech Republic

Funding Acknowledgements

This work has been funded by the Grant agency of the CTU Prague, grant No. SGS10/-291/OHK3/3T/13.

Technical Report Series of DCGI, available at `http://dcgi.fel.cvut.cz/techreps`, Department of Computer Graphics and Interaction, Czech Technical University in Prague, Faculty of Electrical Engineering, Czech Republic.

ISSN 1805-6180

Copyright © Department of Computer Graphics and Interaction, Faculty of Electrical Engineering, Czech Technical University in Prague, December 2012.

Abstract

This technical report summarize results of three-year project which dealt with research in the area of reconstruction and generation of virtual cities. The project connects several fields of research like 3D reconstruction, procedural generation of cities and buildings, grammar extraction, agent technologies, cloud computing, user interaction and further. The unpublished results are presented together with references on published papers. All results and working data were collected at one place and references to this place are mentioned at appropriate parts.

Keywords

procedural generation, 3D reconstruction, user interaction, collaboration, virtual cities, cloud, mobile

Contents

1 Introduction

Several promising research fields were explored in last three years during the ViCiTiS project. The project name was chosen with respect to the main research topic about reconstruction and generation of virtual cities. The main task of the project was to explore and simulate Virtual Cities in Time and Space in distributed environment on different platforms. For this task were developed following research topics: 3D reconstruction, procedural generation of buildings and cities, grammar extraction, cloud computing, agent technologies, user interaction and further.

Several prototype applications from each field of research were developed during the project. Some of those applications and results were already presented on scientific conferences and journals. The rest of created applications were small tests or incomplete works not suitable for conference presentations. This report summarizes all the unpublished and published work and results of the project. The already published work is presented in less detail with references on original publications.

The organisation of the report corresponds to main research fields and project objectives. In Chapter 2 are presented experimental platform for 3D reconstruction and testing data sets created during the project. The procedural generation of cities and buildings is summarized in Chapter 3 and in Appendix A is connected unpublished overview of architectural styles in Europe. In Chapter 4 are described systems enabling visualization of different data types on different platforms and a data distribution. The final step of project was integration of previous parts together. This step was developed only in last year of the project for this reason only preliminary work in the integration is presented in Chapter 5.

2 3D Reconstruction of Buildings and Cities

A prototype application for testing 3D reconstruction algorithms and procedures was developed as essential part of ViCiTiS project. The 3D reconstruction application is called ArchiRec3D . This application is split into several self-sufficient parts communicating by events which brings high variability for reconstruction testing. In the following sections are described main features and principles of ArchiRec3D application. Main idea of ArchiRec3D was already described in two related works. In a paper [SZ12] was presented a 3D reconstruction core, while in poster [STZ12] was suggested 3D recosntruction in virtual environment which is related to section 2.3. We are focused on users who want to re-use the application in this text.

At the end of this chapter are described data sets formed during the project for ArchiRec3D testing.

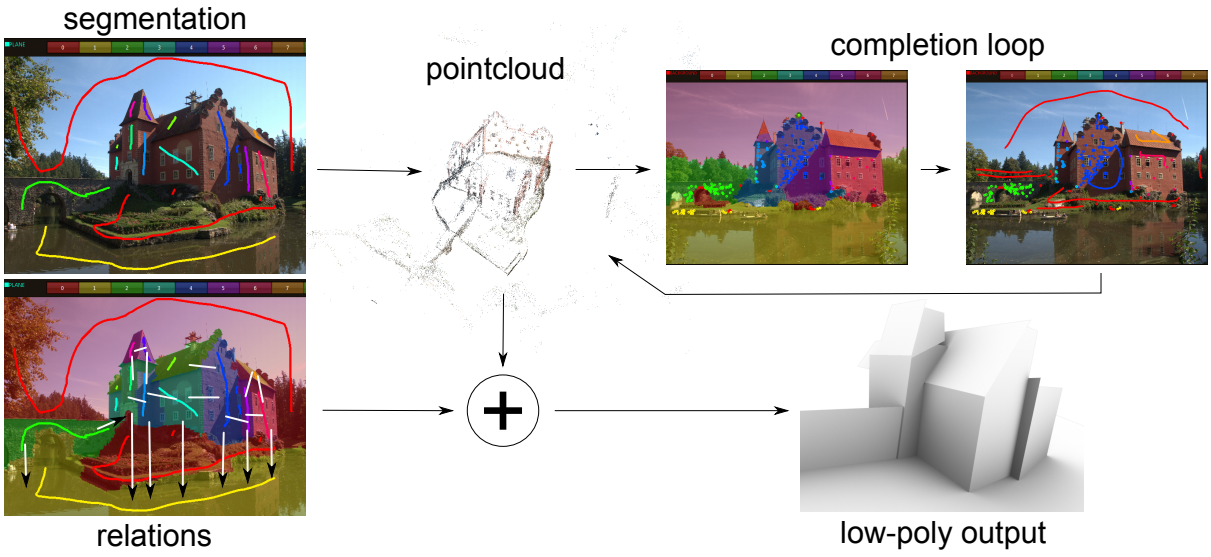


Figure 2.1: Program workflow overview

2.1 Application architecture

The ArchiRec3D application was designed with emphasis on highest variability and separability. Our 3D reconstruction is based on pre-calibrated scenes which can be imported from various formats like is Bundler [SSS06] or APERO [DC11]. The reconstruction algorithms and procedures, see section 2.2, are then applied on imported pointclouds and calibrated photos. A `core` is an essential application part communicating with other application modules which are briefly described in this section. The application binaries and sources are located at

`\\cent\cg\vicitis\3D_rec\ArchiRec3D\`.

We have also implemented import possibilities for other formats like: pointclouds in **ply** or **txt**, geometry **obj**. The final models or pointclouds can be exported to following formats: VRML, X3D, obj.

User interaction - the main user interaction with ArchiRec3D is in 2D view, where user label geometrical objects in the photos. The reconstructed 3D geometry is then visible in 3D view for the posibility of verification. We have implemented several user interactions usable for scene reconstruction and navigation in 3D view, like: store of actual view, annotation of model part, modification of scene origin, measuring in pointcloud and setting pointcloud scale. The user is also informed about state of the application during the work like: display of memory management, thumbnail of chosen photo, statistical informations about pointclouds and geometry.

Application views controls and visualize the main functions of the application. The most interesting of implemented are **2D editor**, **3D view**, **pairs visualiser**, **CG ball synchronization**. The 2D editor is for main user input, see Fig. 2.2 left. The reconstructed geometry is depicted in the 3D view, see Fig. 2.2 right. A special user interface was created for calibration of multi-view system called CGball, see Fig. 2.3. The CGball is a sphere of the normal ball size capturing 6 video streams around the sphere surface.

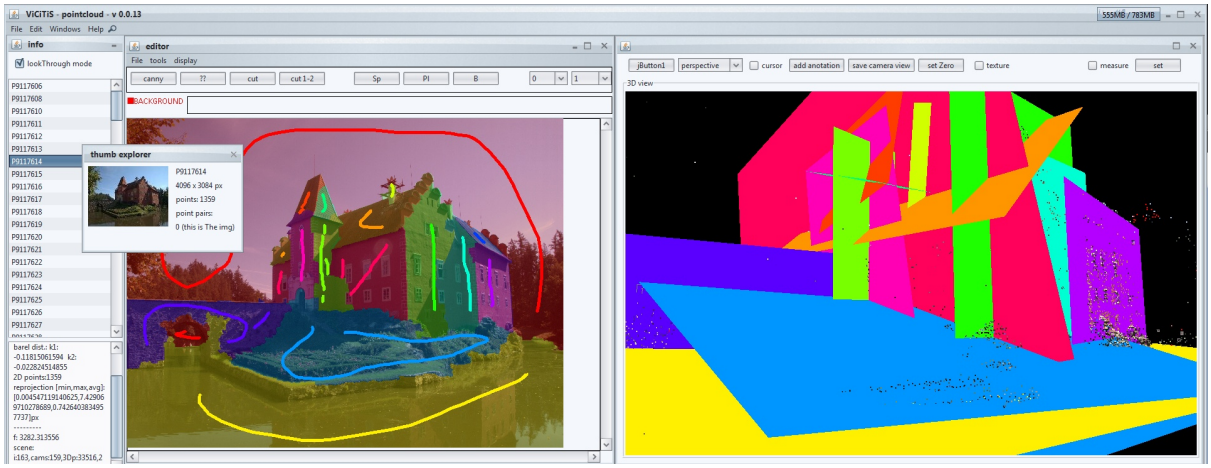


Figure 2.2: Editor and 3D view. The plane primitives (right) are generated based on user defined segmentation (left). The 3D reconstruction is displayed before final step, see section 2.2.

Event model - modules work independently with data in application core and only inform rest of application modules about changes (eg. data loadded, visibility changed,...) by sending event messages. Each module can register for listening for appropriate events and react on them.

Events have three fields (source, actionType, caller). Source is scene object related to event (e.g. point-cloud, mesh, ...). Action type corresponds to what was done with the object, caller

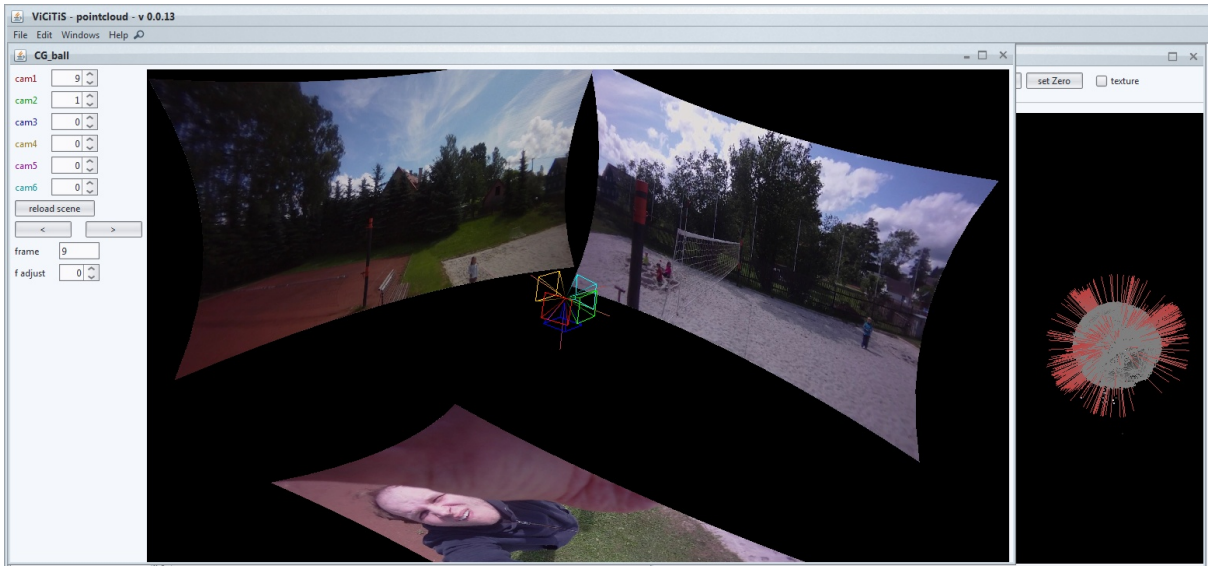


Figure 2.3: CG ball time synchronization

is module responsible for object change. Action type is one of the following (`ADDED`, `REMOVED`, `SELECTED`, `INNER_DATA_CHANGED`, `VISIBILITY_CHANGED`). Modules register as event listeners in `EventRegister` and events are also fired through this class. ArchiRec3D events extends java event model (`java.util.Event*`), all related sources are in package `vicitis.GUI.eventModel`.

With this design, we can react only on changes closely related to the module (e.g. module can update view if point-cloud is changed by segmentation algorithm). This design also enables us to distribute events across distributed networks with connection manager mentioned in section 2.3-Other Systems.

2.2 3D reconstruction

The core of our approach is geometry primitives fitting based on source photo segmentation, see Fig. 2.1. The graph-cut image segmentation is driven by inaccurate user strokes. The image segmentation is then used for labelling of sparse pointcloud points and is also propagated to close photographs. Various geometry primitives are then fitted on labelled points. Thanks to user-defined relations between adjacent geometrical structures, final polygonal geometry is computed. Following algorithms were applied for successfully 3D reconstruction.

Image segmentation is based on well known segmentation technique of finding minimal cut on rated graph. Two algorithms were used. The first one is Boykov Graph-cut algorithm [BVZ01]. The second one is a new implementation of graph-cut algorithm specially designed for regular structures called Grid-cut [JSH12]. The second implementation brings much faster segmentation of input photograph, what is necessary for real-time interaction. The details about graph construction are described at [SZ12], section Image Labelling.

Both algorithms are implemented in C language (original authors implementations) and they are connected to our reconstruction application using JNI technology (Java Native Invocation).

algorithm / image width		640px	2048px	4096px
boykov	segmentation	553ms	15.479s	127.4s
	max-flow	278ms	13.066s	117.353s
grid-cut	segmentation	431ms	7.172s	60.433s
	max-flow	154ms	5.028s	51.652s

Table 2.1: Comparison of Boykov / Grid-cut segmentation algorithm

We have compared both implementations and results are presented in table 2.1. The same user strokes (15 labels) were used for segmentation of differently scaled input photo (the same strokes, the same photo). The whole segmentation time is composed of three consecutive steps: graph construction, max-flow computation, and results reading. The times in the table corresponds to complete segmentation time, that is sum of times of all three steps, and to max-flow time (only the middle step). The graph construction and results reading times are influenced by data shift between technologies (C language ⇔ Java), while the max-flow time is pure computation time of algorithms. The results shows that grid-cut implementation of max-flow algorithm is more efficient than Boykov graph-cut. Based on these results we have set optimal image width to 800px, this attribute can be changed in ArchiRec3D settings file.

3D reconstruction algorithms are used for generation of primary geometry based on segmented photographs and segmentation propagated to the pointcloud. Three different geometry types are created based on user strokes: planar primitives, spheres, and irregular meshes. The algorithms for finding parameters necessary for those geometry construction are presented in following paragraphs.

The **planar primitives** are find using pointcloud points only. The RANSAC algorithm is used to robustly estimate a plane attributes from sparse pointcloud points, if there are 3 points minimally. After RANSAC estimation the finest solution is found using least square while only inliers from RANSAC-phase are used.

Similarly as with planar primitives, the huge **irregular meshes** are generated from pointcloud points (from dense points with normals). The poisson algorithm is used, [KBH06]. All generating points are exported to ply (Stanford Triangle Format) file and external program is called to process them. The mesh is than colored by colors of closest points in the input pointcloud. The poisson reconstruction is called several times with increasing parametres (octree depth and solver divider), this results in gradually refining of mesh geometry. The first mesh is obtained in few seconds (depends on pointcloud size) while the finest solution is displayed after minutes. User is not disturbed by reconstruction process, because it runs on background and only mesh refinements are visible time to time. The sample of reconstructed mesh is in Fig. 2.4.

The **spheres** model parameter finding combines two different data inputs if it is necessary. Due to the fact, that spherical object do not have much features on object surface there is unsatisfactory count of detected 3D points. We detect sphere using combination of RANSAC and Least Square algorithm, if there is enough 3D points, similarly as in planes case. If there is not enough detected 3D points we found center of circle in the input photograph by edge detection in segmented area and than, the sphere radi and center can be estimated only with one 3D point (because is know the line where lies the sphere center and one point on the sphere

surface).

Final geometry estimation combines all generated geometry primitives to form the final geometry. The solution for this step is ambiguous in the general case. So the user define relations between geometrical objects that helps to a decision process to find the output geometry. We have implemented one decision process that works with each geometry polygon locally and the final geometry is generated only on relation inputs. This works fine for simpler objects and it was presented at [SZ12].

The more sophisticated solution can be found using some global optimization method. We have started with implementation of global solution where all primitives are inserted into BSP tree. The BSP tree cells are then labelled by rays projected from camera centres to appropriate 3D points in pointcloud. The final solution is than find by graph-cut. Unfortunately this method does not work yet and we thing that it is the final step for the successful 3D reconstruction.

2.3 Immersive reconstruction

Two systems were developed for the possibility of performing 3D reconstruction in 3D immersive environment. Both systems displays the 3D view in stereo, the rest of the application is displayed unchanged. The first one is an application native support described in the following section. The second one is a synchronization protocol enabling scene editing or viewing on other platforms, see section 2.3.

Native support

The reconstruction application can be switched to stereo support in dialog called Anaglyph settings, see Fig. 2.5a. Although the dialog is called anaglyph, another stereo types are implemented (passive mode with polarization glasses, and active mode with shutter glasses). Following anaglyph types are implemented: red-blue, red-green, red-cyan (called gray), and coloured red-cyan with following optimizations (full-color, half-color, optimized), see [Wim07]. The 3D support is based on java Anaglyph Canvas3D available online at `http://sourceforge.net/projects/anaglyphcanvas3/`. The stereo support is disabled by default and can by enabled in `settings.xml` by setting field `use3DOutputDevice` to `true`. Example of 3D anaglyph view in compare with normal 3D view is figured out in Fig. 2.4.

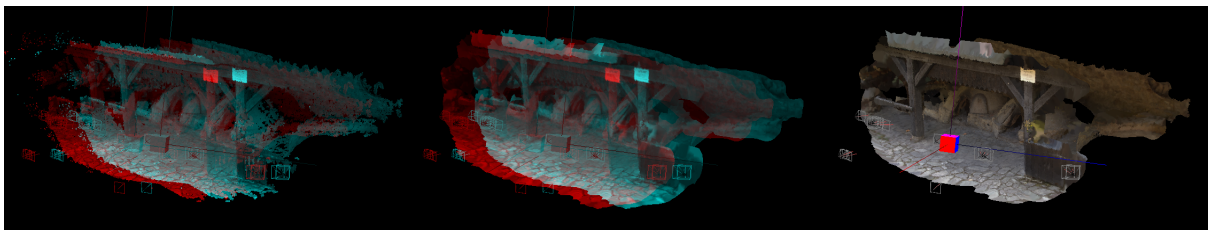
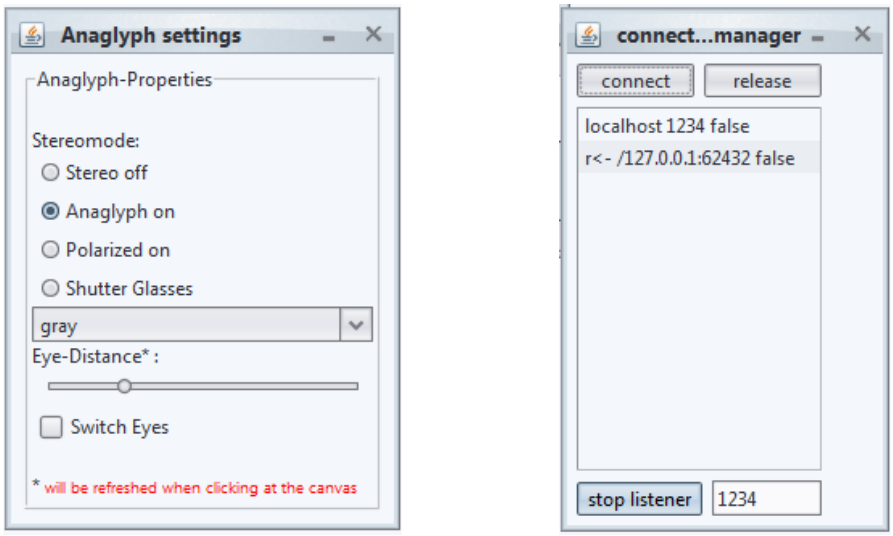


Figure 2.4: 3D reconstruction with stereo-view (anaglyph - gray). Input 3D pointcloud (left) is reconstructred using poisson reconstruction (middle). Final reconstructed mesh is colorized by input pointcloud, figured in true colors (right).



(a) (b)

Figure 2.5: Dialogs: (a) Stereo settings; (b) Connection manager.

Other systems

A fast synchronization protocol was designed to port the 3D reconstruction to another platforms.The idea is that lightweight client is developed for each specific platform and it has response for some parts of the application. The 3D view client was implemented for CAVE platform used in Institute of Intermedia (IIM) where we have done testing and it was implemented in java also. So we have the possibility to display 3D stereo view on the reconstruction computer, any other computer with Java3D support (streo wall) or in a CAVE build on `cavelib`, see Fig. 2.6. The connection to clients can be established in connection manager (Fig. 2.5b). We suppose, that clients are listening on an open TCP port waiting for data send by the reconstruction application. The application does not control what or how the clients displays the data because the application does not know anything about clients possibilities. The communication protocol is described in section 5.2.

CAVElib implementation

The support for 3DRec protocol in CAVE system was implemented as a module for `cave_framework` - lightweight middleware for developing applications deployable in a CAVE system. `cave_framework` can be used with various backends and displayed in a CAVE system (with `cavelib` backend) as well as at common computer (using `SDL` or `GLUT` backends). Therefore, our implementation can use the same variety of display devices. Currently, the only application using this module is `objshow` (available as a part of `cave_framework` distribution). There is a support for displaying multiple modules either at the same time time or as a switchable entities. This enables us to visualize multiple remote models at the same time. We have currently implemented messages related to pointclouds only.

Usage

The simplest command to display the model is as follows:

```
objshow --model vicitis://X
```

In order to specify port to listen on or size of points for display, parameters `port` and `point_size` can be used as follows:

```
objshow --model vicitis://X[port=1234,point_size=0.1]
```

Source code for cave_framework is currently available only upon request from Institute of Intermedia. It will be published at IIM website in a near future.

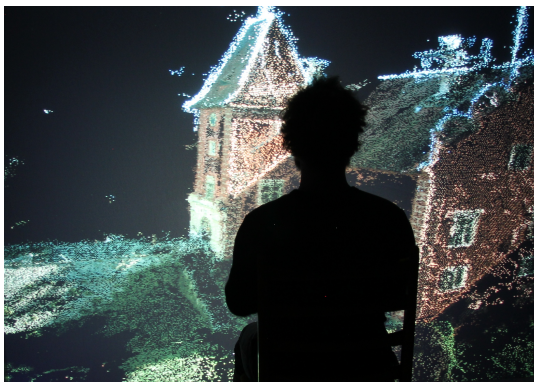


Figure 2.6: 3D reconstruction in CAVE - stereo disabled for better visualization.

2.4 Data sets

The ArchiRec3D was tested on several data sets created during the project. Those sets are composed of calibrated input photographs, sparse point-clouds and dense point-clouds. Sets were mostly calibrated by bundler [SSS06] and dense point-clouds were generated using PMVS [FP10]. All data sets are located at `\\cent\cg\vicitis\3D_rec\data_sets\`. Data sets are related to city 3D reconstruction but vary in the reconstruction subject and detail. They can be separated by geometry type into regular structures (like buildings) and free form objects (like statues). Tables 2.2 and 2.3 summarizes sets.

We have also prepared two public sets for another scientists interested in 3D reconstruction and model creation. The first one is an extension of previously mentioned data sets. It contains several buildings around the city of Prague. The biggest set is composed of 20 parts covering the Charles square in Prague. All buildings around the place and a few statues are captured in this set. Detailed description of this set containing more then 20M points is at `\\cent\cg\vicitis\3D_rec\data_sets\_www`, see Fig. 2.7b.

The second public set are photos of the Langweil model of Prague. The Langweil model is a paper miniature model of Prague from 18th century created by Antonin Langweil. The model itself is located in the Museum of Prague. We have selected three parts of model and make them public available after registration, see Fig. 2.7a. The data set was presented during

name	photographs count	sparse point-cloud size	dense point-cloud size
amadija	414	121k	2.2M
amadija_cupola	59	51k	430k
cervena_lhota	159	33k	611k
doudleby	83	53k	1.3M
faust_house	146	67k	1.5M
front_facade	9	4426	547k
house_hubertus	36	21k	380k
jordansky_sarkofag	8	5k	340k
computer_graphic book	7	653	344k
saalburg_gate	59	72k	600k
saalburg_oven	18	3303	260k
slany	597	440k	17M
srni	48	14k	400k
radnice_gocar	43	10k	951k
vez_na_vltave	40	1502	64k
zeleny_domcek	16	1643	90k

Table 2.2: Overview of data sets, regular structures like buildings.

name	photographs count	sparse point-cloud size	dense point-cloud size
purkyne	35	2496	47k
saalburg_stone	26	16k	152k
socha_kuks	3	365	36k
telc_kalvarie	37	13k	482k

Table 2.3: Overview of data sets, free-form structures, statues mostly.

poster session on Eurographic2011, see [SZ11] and detailed description can be found in technical report [SBZ12].

(a) (b)

Figure 2.7: Public data sets: (a) Three parts of Langweil model; (b) Charles square dataset.

3 Cities Procedural Generation

Two main fields of procedural generation were studied during the project. The first was generation of whole cities which is described in the following sections. The special part of cities generation is a generation of infinite cities, described in section 3.3 where our approach to infinite cities generation is described in section 3.4. The second field of our study was procedural generation of buildings and grammar derivation from existing buildings, see section 3.5. In section 3.6 are summarized results and outputs of our work in this field.

The detailed overview of existing approaches to cities and building generation was presented at [DSŽ11], together inverse approaches used in computer vision areas where models are found in images based on defined grammars.

3.1 City modeling approaches

There are two main approaches to procedural city modeling – behavioral and geometrical. However recently researchers succeeded in combined both approaches together.

3.1.1 Behavioral city modeling

Behavioral approach focus on simulation of city development in time. The simulation starts with a city layout acquired from a map of a real city o from a artificial one. It is important to know the type of the building, because it has a significant effect on the 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, only few behavioral modeling system can work with arbitrary oriented buildings. Visualization of the city is not usually too much 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 usualy not considered during the modeling.

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

3.1.2 Geometrical city modeling

The geometrical approach focus on creation of a visually pleasant 3D city model that do not evolve in time. This approach is usually used for computer games that need a static city model. Realistic look of the city is the primary goal in geometrical city modeling. The modeling systems use a heightmap and can have arbitrary oriented buildings. The output of the modeling software is usually a detailed polygonal city model with textures.

3.1.3 Combined city modeling

In recent years some researchers try to combine both previous approaches together. In 2009 Vanegas et al. [VABW09] presented a system that combines both geometrical and behavioral

approach in city modeling. Their approach combined both city evolution in time and detailed realistic 3D look of the city. Weber et al. [WMWG09] presented in 2009 a system, that can quite precisely predict city development from a city map or create a completely new city.

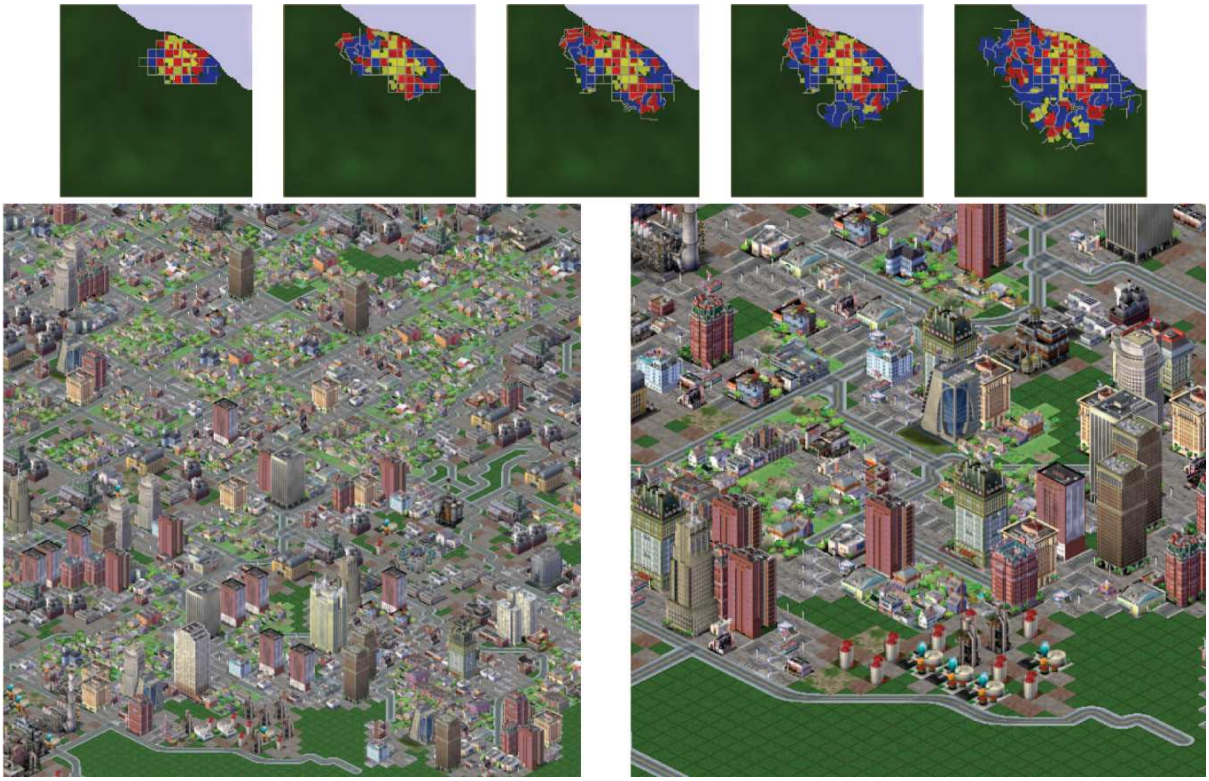


Figure 3.1: An example of behavioral city modeling published by Lechner et al. [LWWF03] (top) and its visualization using SimCity 3000 (down). SimCity 3000 is a comercial computer game developed by Maxis in 1999. Different colors represent different building types/land utilization.

3.2 City modeling workflow

This section presents the most common workflow for city generating based on street network. This approach was for the first time presented Parish et al. [PM01] in 2001 and now it is used in nearly all works concerning geometrical procedural city modeling.

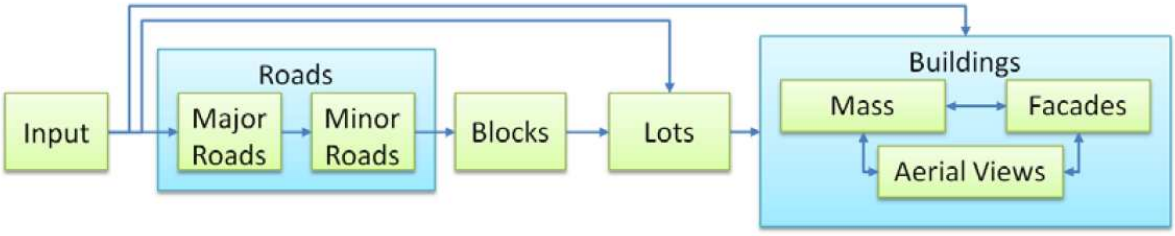


Figure 3.2: General pipeline for city modeling. [VAW+10]

The key part in the workflow is the city road network that is modeled first. The road model is created using a L-systems [PLH+90] technique extended by context sensitive rules[PM01]. The L-systems were originally developed for procedural modeling of plants, but they can be used for road network modeling as well.

In 2006 Muller has proposed a new language called CGA (Computer Generated Architecture) [MWH+06] that can be used 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 procedural building modeling area, because the previous modeling techniques were suitable.

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

- **Major roads** usually represents highways or other big roads. In some modeling systems these roads must be placed manually. Creating the major roads is the first step in the city modeling workflow.
- **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.
- **Minor road** is a road inside a quarter used for subdividing the quarter. Minor roads are usually generated by contex-sensitive L-systems.
- **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.
- **Building lot** is a land area dedicated for generation of single building.
- **Building mass modeling** creates rough geometric shape of the building. A building mass is usually generated by CGA grammar rules.
- **Building facade modeling** splits building to levels and creates wall geometric. Building facades is usually generated by CGA grammar rules.

3.3 Generating Infinite Cities on Regular Grid in Real Time

In 2001 and 2003 Greuter et al. presented an approach to procedural generation of infinite cities in real-time [GPSL03c, GPSL03a]. In his paper the infinite city consists of a regular rectangular grid of building lots with single building on each block (see fig. 3.3). 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. 3.4). This number is used as a seed for the pseudo-random building generation of that building lot (see fig. 3.5). The generated buildings are saved into cache to save system resources.

Figure 3.3: Previous approach in infinite-city rendering published by Greuter et al.[GPSL03b, GPSL03c] displayed from the street level – real-time rendering, generated online; Note the regular rectangular shape of the street network. The regular rectangular grid of building lots does not make the generated city look much realistic.

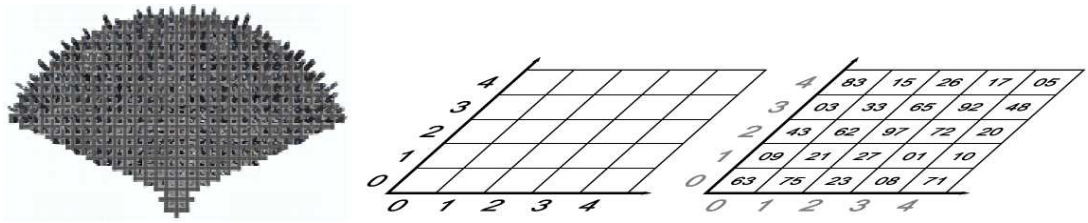


Figure 3.4: The rendered buildings are determined using intersection of the viewing frustum and building bounding boxes.(left) A hash function assigns a seed for the pseudo-random building generator to each building lot according to its coordinates.(right) Source:[GPSL03a]

3.4 Our approach to Infinite Cities Generation

We have presented a novel technique for generation of pseudo-random infinite cities in a real-time [DZ11a]. The generated cities can have arbitrarily oriented streets and building blocks can be arbitrarily shaped. The shapes of city buildings are determined using a pseudo-random generator that uses building coordinates as the initial generator seed. Our appearance of an infinite city looks more realistic than in previous approach made by Geuter et al.[GPSL03b, GPSL03c] (compare figures 3.9 and 3.3).

To verify our approach we have implemented a Silverlight application (.Net equivalent of a Java applet) that interactively generates infinite street networks according to generator parameters. The generated street network can be imported with other tools used for city modeling like CityEngine[Ers] (see figure 3.8) and converted to polygonal model (see figure 3.9). The buildings block are pseudo-randomly subdivided according to Paris and Müller[PM01]. For generation of building geometry we used pseudo-random procedural approach based on grammars that was described by Müller et al. [MWH$^+$06].

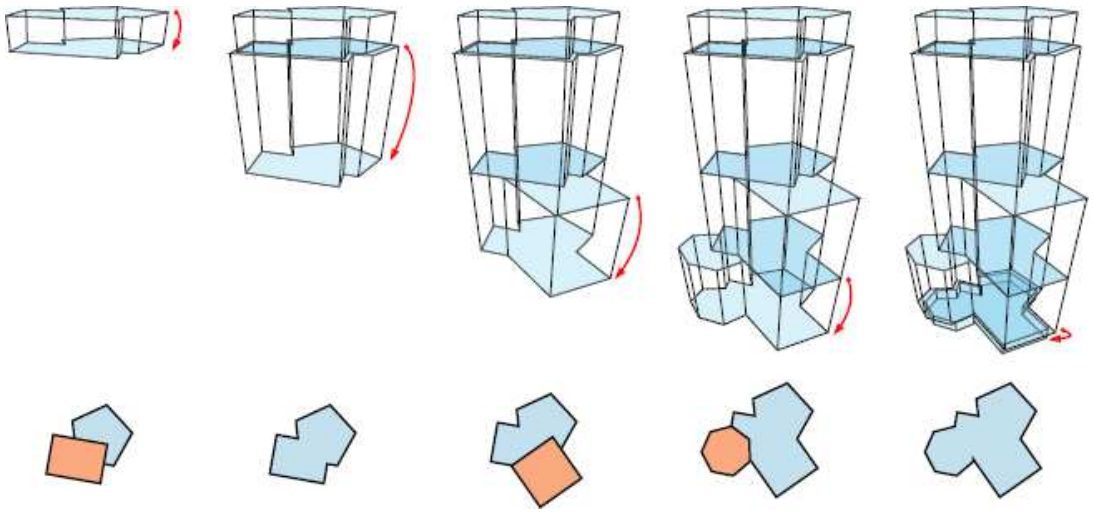


Figure 3.5: In the method of procedural generation of buildings proposed at [GPSL03a] the building is generated from top to bottom. The roof of the floor consist of two random primitive shapes. The top floor has the shape of slightly enlarged roof. Each next floor adds one random primitive shape to its shape. At the end of the process the bottom of the last floor is slightly reduced. Created building has to be rescaled to fit into its building lot.

We have created an algorithm for generation of infinite pseudo-random non-periodical arbitrary-oriented street network that can run on-line in real-time.

3.5 Grammar-based Building Modelling

In computer architecture, grammars are generally used in one of two capacities: Building generation and building reconstruction. These are two opposite, but complementary approaches which both exploit the grammar's capacity to describe a sentence of simple terminal symbols in terms of their structure and hierarchy. Using a *parser*, we may then derive a structure from a string of terminal symbols, or conversely, using a *generator*, we may repeatedly apply production rules to create the description of a building satisfying some parameters we choose.

The most developed example of the first application are the CGA grammars of [MWH$^+$06], which follow the classical workflow of building generation, from building mass to building facades. The shape grammar is an attributed grammar operating on shape and their spatial context, the description of which is in turn used as the terminals of a second attributed grammar, which allows the generating engine to generate buildings exhibiting variety based on the values of pre-determined parameters. In effect, both grammars are used to describe a set of restrictions we place on the shape of the building; the shape grammar, describing basic operations, makes sure that the building shape remains locally valid, while the higher-level grammar, restricting the sequence in which these operations are applied, enforces the particular style or shape of the building.

An example of the second group of grammars is [VAB10]. This so-called *Manhattan-world Grammar* is used as a building description in image-based building reconstruction, wherein computer vision algorithms are used to determine the parameters of this attributed grammar to match a photographed building as closely as possible. In this application, a fixed grammar is a structured way of representing a set of assumptions we accept about the shape of the building we are trying to reconstruct.

While the mechanics of building descriptions are different in both of these cases, there is a clear common purpose – to restrict the generated shape so that it represents a valid building conforming to our requirements, while providing sufficient flexibility so that variety can be

Figure 3.6: Panorama of an infinite city generated in real-time using method [GPSL03a, GPSL03c]

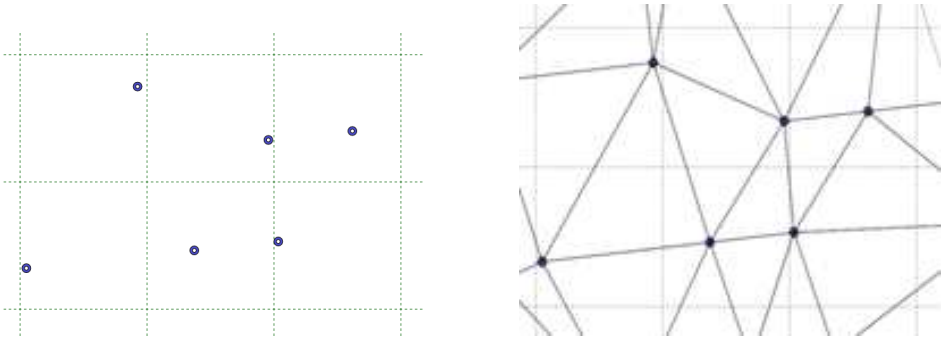


Figure 3.7: Our approach - Left: Randomly generated points in an infinite grid; Right: Delaunay triangulation that forms the streets

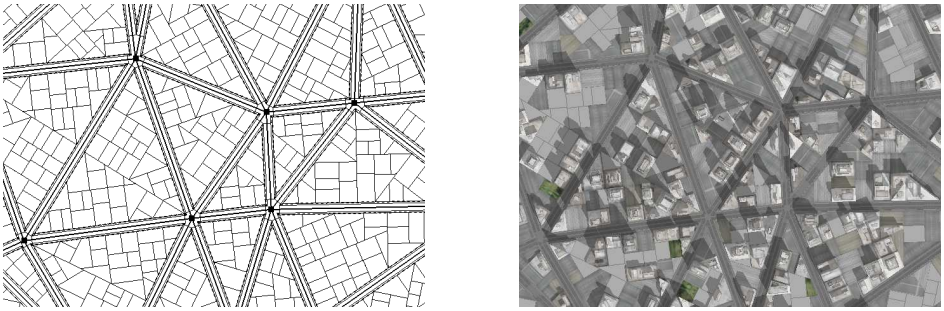


Figure 3.8: Our approach - Left: Building blocks/lots generated according Parish and Müller[PM01]; Right: Generated city - view from above

accounted for.

The reason why the higher-level CGA grammar, as well as Manhattan-world grammar, are attributed is so that a variety of buildings may be created, either by modifying the sequence of productions or by tweaking the parameters of geometric operations. In Cityengine [Ers], this property of CGA grammars is exploited by allowing the generation of different building via randomization of paramteres within certain allowable ranges. The price for this, however, is that the grammar has to be carefully constructed so as not to allow invalid building configurations, which could be incurred by selecting parameter values not expected by the author of the grammar.

In order to produce a variety of plausible buildings, a higher-level CGA grammar first has to be derived. While the original paper describes a very straightforward way to define a building from the ground up, it is much more complicated to define a grammar capable of generating an existing building. Indeed, defining a suitable Manhattan-world grammar is in fact one of the key contributions of [VAB10]. In our experiments, we have found that deriving such a description is very work-intensive and a trained user (as in [Vam11] or [DZ11b]) can take days or even weeks (depending on the complexity of the building in question) to derive such a description, even when consulting with profesional architects.

Generalizing such a description is an even more formidable task. Even if [1] has met with some success in doing so manually, the results were deemed unsatisfactory by architectural experts.

[1] citovat Eleniny goticke budovy

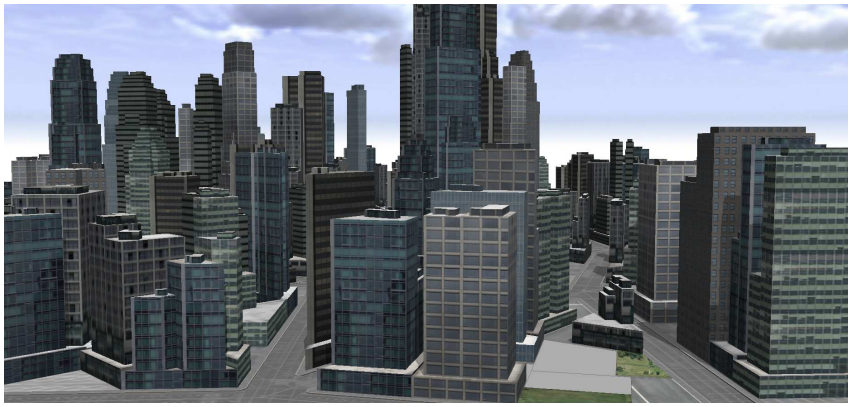


Figure 3.9: Our approach displayed from the street level – real-time rendering without scene post-processing, generated offline

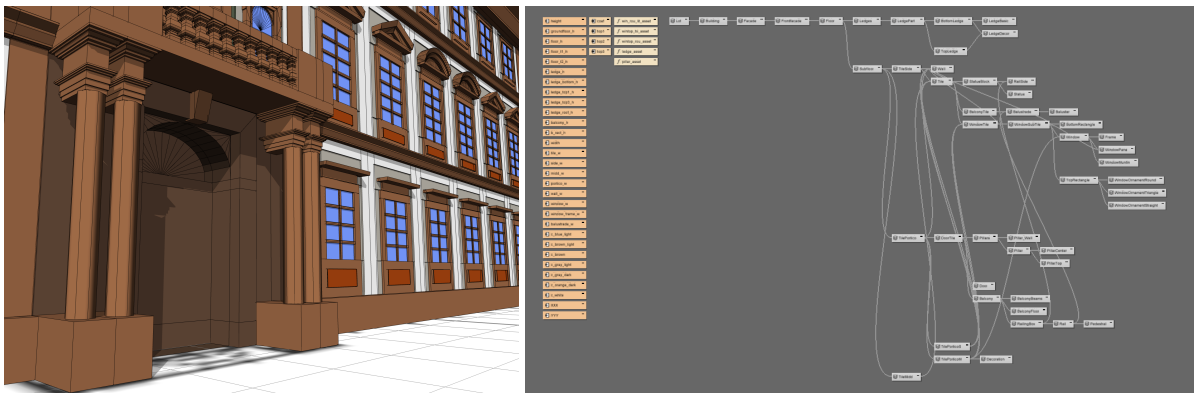
The most significant problem appeared to be that the criteria which experts use to determine conformity with and architectural style and stylistic plausibility are very vague and remained elusive even after much consultation with experts.

It is tempting, then, to attepmt to produce a more general grammar by abstracting from grammars describing individual buildings. The structure of a grammar lends itself well to such merging, as by defining correspondencies between non-terminal symbols, we may merge their respective production rules and thus arrive at a new grammar defining the entire group of buildings. Unfortunately, even having solved the task of automatically defining these correspondencies and parameter mappings, such an automatic process would lack a significant amount of semantic information, determining eg. whether two possible values of a parameter define a discrete or a continuous range, or if there are cases where corresponding non-terminals are not in fact interchangeable. Until there is either a set of specific formal criteria the generated building has to satisfy, or the degrees of freedom within an architectural style are formally described, this taks remains impractical to solve with any useful result.

3.6 Outputs of our work in Grammar-based Building modelling

We have started with test-case example which should show us that two buildings sharing similar properties of origin like is a time era, geographical area, and architectural style should be generated by similar grammars. The selected buildings were Tuscany Palace and Thun-Hohenstein Palace in Prague build in Baroque style. Two rule files in the CGA grammar were created, each generating corresponding building. We have tried to to catch and keep architectural style specific to this kind of buildings when creating a hierarchy of CGA rules. The resulted grammars then have similar shapes that can be measured, compared and evaluated, see Fig. 3.10. For comparison of our two palaces, it is possible to find the same parts and also differences in their hierarchies. The test study prove the idea, but also shows the weaknesses of the approach. The credibility of this approach can be proven only by creating more generating grammars from wider building spectrum of one style. See [DZ11b], for deeper details.

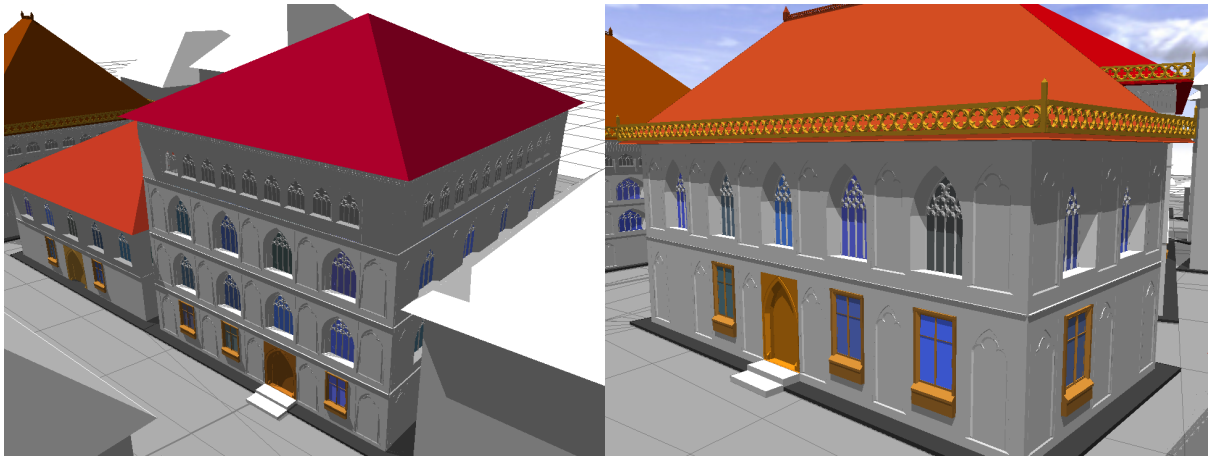
The results of previous work shows us that if we want to continue in this direction, we have to choose simpler architectural style (not so visually rich as Baroque style), create more grammars, possibly automatically and find an automatic comparison algorithm. As the simpler



(a) (b)

Figure 3.10: One of generated baroque buildings and generating grammar hierarchy.

architectural style was chosen Gothic style, see architectural study at Appendix A. The study shows us the most common properties of the style. The selected building elements of style were modelled and used to create several grammars, see Fig. 3.11. Unfortunately we were not able to find the ways how should be solved the automatic solutions and we have finished work on this task after second year of research.



(a) (b)

Figure 3.11: Examples of generated Gothic buildings.

4 Collaborative visualization on different platforms

4.1 Collaborative environment using cloud

During our effort we have presented a project on building interactive multi-user 3D services in the Windows Azure cloud[DK12]. The future goal is to build a prototype platform that would allow creating 3D virtual environments stored in the cloud and accessible by a variety of devices with very different performance and capabilities. Some of the devices (e.g. mobile phones) may not even be able to store the entire scene geometry in their memory and will have to use only partial content downloading. The cloud-side application will automatically and instantaneously adjust and communicate the evolving world content to reflect the actions and changes effected by the many users interacting with objects in the 3D environment.

During the implementation we have focused on new Microsoft technologies. For the mobile client we use mobiles phones with Windows Phone 7 operating system. Rendering of the shop is based on a combination of Silverlight and XNA. The cloud service is implemented using the Windows Azure platform and is accessible via a desktop Windows PC or a device with a different operating system using a browser with Silverlight 5.

To reuse the rendering code between the web browser with Silverlight 5 plugin and Windows Phone 7, we have created a library that encapsulates the differences between those platforms. The most challenging issue of the library was with the graphics shaders. Windows Phone 7 forbids using a custom vertex and pixel shader, because it runs on a device with a graphics chip that does not support shaders and thus only a set of pre-defined shaders can be used. On the other hand, Silverlight 5 3D does not have some high-level rendering functions to save downloading size and achieve higher portability. Those functions have to be implemented using low-level vertex and pixel shaders. Thus Silverlight 5 forces its developers to use shaders.

To overcome the shader problem we had to transfer the pre-defined Windows Phone 7 shaders to Siverlight 5 and implement Windows Phone 7 high-level rendering functions using low-level Silverlight 5 shaders. In our applications we have to use only those functions available on both platforms, thus we cannot use other shaders than those that are pre-defined. In March 2012 we plan to provide the XNA Interface library to other developers to ease them developing 3D Silverlight projects for multiple platforms.

Our first prototype of a 3D environment is the 3D Teapot application, a 3D object that can be manipulated (i.e. changing orientation or color) and observed in real-time by multiple users simultaneously, using different terminals. The teapot representation is being stored on Windows Azure. A Web role is used as frontend, while the actual 3D Teapot state is being permanently stored in Azure SQL and updated with every manipulation of the teapot.

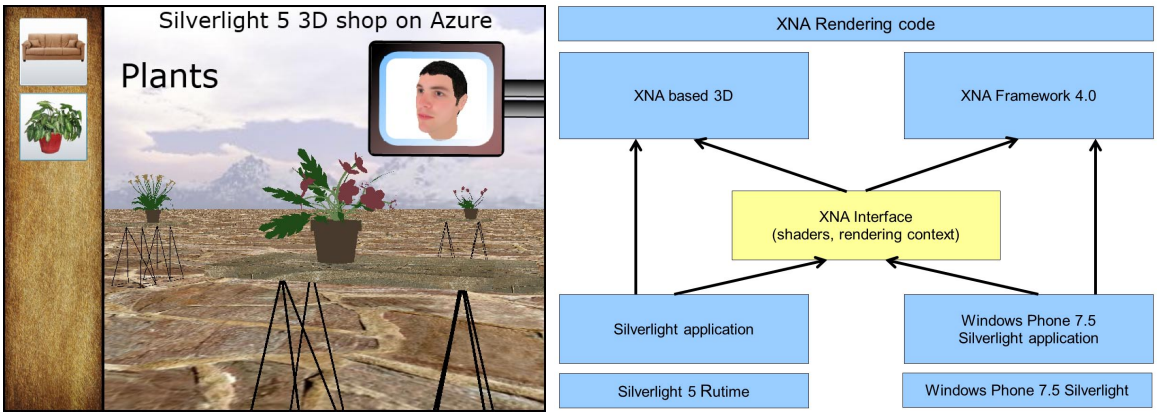


Figure 4.1: Left: 3D e-Shop with Talking Head on Windows Azure; Right: Code reuse, browser and phone versions

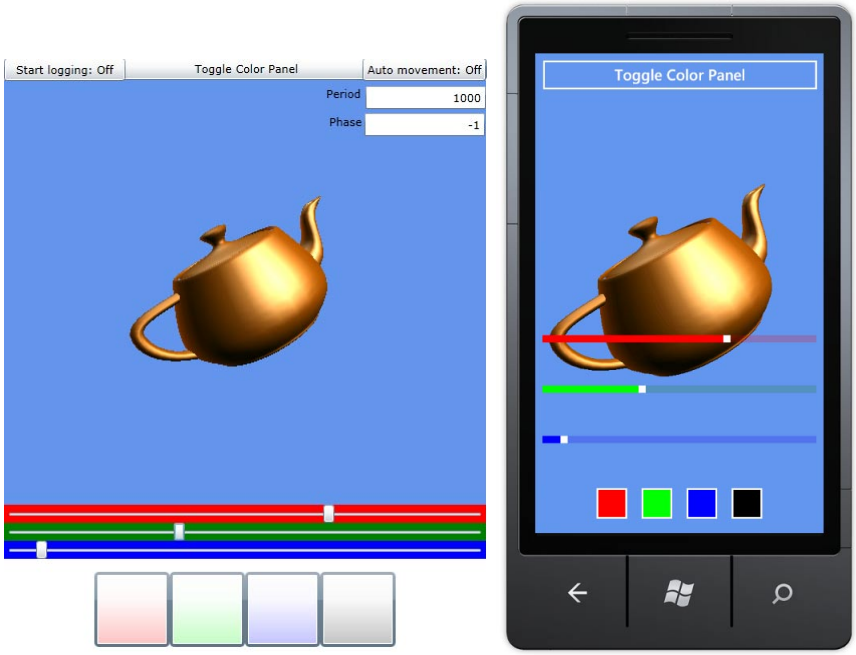


Figure 4.2: Screenshots of two client versions of the synchronized Windows-Azure-based 3D Teapot, accessible simultaneously by multiple clients: mobile (XNA-based) and desktop (Silverlight 5 in a browser).

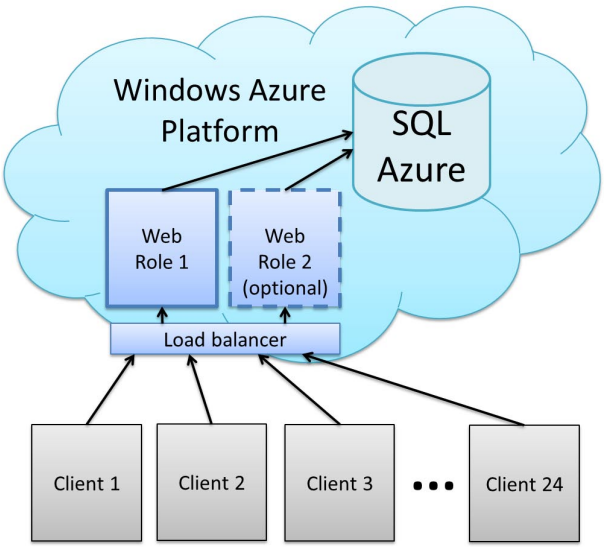


Figure 4.3: Architecture of the experiment: Up to 24 client instances were simultaneously trying to connect to the web service to synchronize state of a single object in the database every 100 ms.

Figure 4.4: Tests of multiple parallel clients simultaneously attempting to manipulate an identical 3D Teapot stored in Windows Azure cloud.

4.2 Remote collaboration in a CAVE system

We also took another approach to collaborative work, aimed mostly for use with immersive systems like the CAVE[CNSD+92]. This approach is based on the idea of sharing the visualization instead of the source data. We proposed and used this architecture in the project CAVE to CAVE (C2C, 2008 – 2010) [BTH+09, BTH+10].

The basic idea is to get the visualized content from the local system and send it to remote client(s) as a video stream. We focus on getting as-low-as-possible latency and as-high-as-possible video quality. In order to be as universal as possible, our soluton aims to be independent on the application used and doesn't require any modification of the application. The streamed content can be accompanied by bidirectional audiovisual data streams from a video cameras to provide the user with an video conferencing and enable discussion about the content. We have also implemented a remotely controllable virtual input device (section 4.2.4) to allow the remote user to control the visualization. The whole system was demostrated on several international conferences (Terena Networking Conference 2011, Cinegrid Workshop 2011) and described in [UTZH12].

System overview

For the purpose of sharing visualizations from our CAVE system a framework for video processing and streaming has been developed. It provides means for image acquisition, color and format conversions, compressions, decompressions and transportation. The main goal was to archive as-low-as-possible latency while maintaining high throughput. The first version of this framework, developped for project C2C, was extended and enhanced for this project. The core of the framework was rewritten from scratch. This section describes the framework in it's current state. Some of the features were originaly written for C2C version of the framework and then ported to the current version during this project. The framework is called `libyuri`.

4.2.1 Design of libyuri

Video processing pipeline usually consists of several steps that are mostly independent on each other. By exploiting this feature we can run the processing steps for different video frames simultaneously and thus increase throughput, provided the underlying computer system can effectively run several threads of execution at parallel. This is true for most of current CPUs which have often 2-4 cores allowing to run 1 or 2 threads simultaneously at each of them. This approach also minimizes performance impact introduced by processing of stereoscopic signal. On the other hand, the need to hand the data from one processing step to another may introduce additional latency. To avoid this we separate the steps into threads instead of separate processes, so we can keep data in common memory space. We also use smart pipes connecting the outputs and inputs of processing steps that can be tuned to either force processing of all data or to drop some frames to lower latency when the procesing pipeline is overwhelmed. The pipeline consists of several processing nodes representing the processing steps. Each of the nodes can have several inputs and outputs that are connected to other nodes and they form an oriented graph of the processing pipeline. In our framework the graph is described by an XML file defining all the processing nodes and links between them. The XML file is processed at runtime and the pipeline is then created automatically.

4.2.2 Data acquisition

In order to share data from a running CAVE system (or any other visualization source), we need to acquire the data being displayed. As we aimed for approach usable independently on the application used and not requiring modification of the application, we focused on methods working unintrusively. We have explored several options:

Acquiring data directly from an input of the visualization device (projector, monitor) by grabbing the signal from video bus (typically VGA, DVI or HDMI). This method has great advantage of being completely transparent to the visualization application and does not affect the running application at all. It also provides stable frame rate of the data acquired. On the other hand, it has some disadvantages that are very limiting for our use: This method needs to grab whole screen, which makes it unusable for visualizations not utilizing whole screen, like windowed applications. The acquired frame rate is stable, but usually different from the rate at which the application produces frames, thus it may lead to some unnecessary processing of the same frame several times or to skipping some frames. It also requires the visualization system to be synchronized to vertical blank, otherwise tearing artifacts may appear in the acquired data. And finally, typical grabbing devices operates on the frame rates usual for common display (60-80 Hz) and thus can not handle active stereoscopy.

Acquiring the content of the framebuffer. When the visualization application prepared video frame for display it stores it into a frame buffer and notifies the system that it is ready to show the image (to swap buffers in double buffer configuration). At this moment we can read the data from the frame buffer and send them for processing. Our implementation uses this approach by wrapping the application and hooking onto the call to swap buffer call [TB10]. Advantage of this approach is that we receive exactly the amount of data the visualization application produces without any duplicates. It is also very easy to process quadbuffer stereo this way. By hooking onto the calls to setup the projection, we also know the position and dimension of the application window, so there is no issue with windowed applications. Another feature is that we can get the content of Z-buffer and thus having the knowledge about the depth of the scene (ie. for generating new views not rendered by the application). The main disadvantage is that the data acquisition has to be run in the rendering thread of visualization application, so it impacts speed of rendering. Also, the processing is done on the same system as the visualization, so it has to share resources (CPU, memory) with the visualization and may lead to another slowdown of the application. The latter problem can be solved by setting the CPU affinity for the visualization application and for the processing to disjunkt groups of CPU cores.

Recording the calls to the GPU and the replaying it and re-rendering (as in [EF07] or [HEB+01]). This approach assumes that we can record all commands the application sends to the GPU (or to the driver of GPU) and then replay the command sequence elsewhere to produce the same image. The biggest advantage of this approach is the ability to get the video data at different (possibly much higher) resolution that the visualization application produces. To some extent it is also possible to get stereoscopic video data even though the application renders only monoscopic images. The biggest disadvantage is very variable data rate and difficult handling

of actual data associated with the commands being sent to the GPU. Also the re-rendering process may be very resource intensive.

Currently the system is ready to use the first two approaches. For our applications using quad buffer stereo and active stereoscopy it is more suitable to use the frame buffer grabbing approach.

The system is also able to acquire data from many different sources, mostly video cameras by using off-the-shelf video capture cards and standard interfaces like IEEE1394 or USB.

4.2.3 Video streaming

After acquiring the visualized data, we have to stream it to the remote client. In past we used several methods using the RTP protocol (with either raw video or a MPEG2 compressed stream). The current version is mostly focused on communication with a device `MVTP-4k`[HKU+10] developped in CESNET. The primary streaming format is compatible with the format this device uses and audiovisual data can be streamed to and from this device. This allows very high video quality (currently HD and 4k resolutions) and low latency. On the other hand, this approach has very high bandwidth requirements (500Mbps – 1200Mbps per single video stream).

4.2.4 Remote interaction

In order to distribute interactive content we also need to provide means not only to get the audiovisual content to the remote side, but also to provide the remote user with means to control the content generation. For this purpose we developed system consisting of simple data acquisition performed by a python script and a linux kernel module providing virtual input device that duplicates the actions of remote user in the local system. For transport of these data we can use any data connection between the the systems, usually we use either a ssh or raw udp connection (provided by a tool nc).The kernel module has following parameters:

debug Setting debug to 1 enables debug output to the syslog.

local_bits Set to 32 or 64 to specify bit width of the local system.

remote_bits Set to 32 or 64 to specify bit width of the remote system.

Usage of vev module

The module creates two files in the `proc` filesystem for data input.

/proc/vev/feed File for inputing raw events directly from `/dev/input/event*` device. It respects the values set in `local_bits` and `remote_bits`.

/proc/vev/raw File for inputing input events stripped of the timestamps. This requires script `evstrip.py` to be applied to the raw data from `/dev/input/event*`. It is independent of the bit width of local system.

The module also creates new input event interface for the virtual device (usually stored as `/dev/input/eventX`, where X is some number assigned by the kernel). Events generated from this device are copies of the events written to the module's `/proc` interface.
The source for the kernel module `vev` and the python scripts are available from: `\\cent\cg\vicitis\Vev\vev-0.6.tar.bz2`

4.2.5 Agent-based virtual network

The above methods were demonstrated at several occasions and received quite a positive feedback. One of the challenges for each of this presentation was the actual setup of the whole system. The configuration needs to by manually adjusted to specify correct network addresses, ports and other location specific configuration options. Also the quality of the video may need to be adjusted for the quality of network connection available. Another task is to build appropriate processing pipeline graph for the application being used as well as to setup any processing of the video data, if some is needed. All of this is time consuming and slows down deployment of the system at new locations or for new applications.

Our proposed solution to this problem is agent-based virtual network connecting all the users and automatically configuring the data acquisition, processing, transmission and adaptations. It also should serve as a QoS monitor with the ability to dynamically reconfigure the transmission if the quality of the network is not sufficient. Each of the users or devices that are used in the system should be represented by an agent. All the agents are aware of the other agents a periodically check the underlying network for any problem on the communication route and re-routs the traffic when needed. The agents use several methods to find others and in most cases should automatically connect to the rest of the virtual network without any user interaction. The only configuration they should need is the description of their local system, so they can configure applications and the adaptations. Application specific settings can be stored locally in the agent configuration.

The idea of an agent-controlled system was already proposed in C2C project and basic agent node was implemented. During ViCiTiS project, we partially implemented the negotiation of the content and tested the automatic generation of processing pipeline and starting the transmission.

5 Integration

5.1 CityEngine plugin

One of the project goals is to design and implemented plugin for the CE, which allows to control the software from the network. It allows you to connect to it from a remote station and take advantage of CityEngine tools from distance in real time. The user can also remotely set grammar rules for desired building models.

For implementation of the plugin we choose Jython scripting language that is integrated in 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 generation 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.

5.2 3D reconstruction in Virtual Environment

An environment for multi-user immersive 3D reconstruction was described in Eurographic poster [STZ12]. The poster present possible solution how the 3D reconstruction can be performed in immersive 3D environment like CAVE or stereo-wall using tablet PC and control computer. Fundamental layer of proposed approach is fast lightweight communication protocol designed specially for synchronization between different platforms (tablet - Android, 3D recosntruction control computer - Java, CAVE realtime rendering - C++). The protocol described in following section was fully implemented and tested for Java and C++ language within the project, results were presented in the poster. The poster task of distributed reconstruction platform was not been developed, so the protocol contains only actions necessary for visualisation purposes and need to be extended for this. Figure 5.1 shows two instances of reconstruction application while the first one is used for editing, the second one is used for model verification from other viewpoint, data between instances are automatically synchronized.

Communication protocol

Simple protocol is based on TCP communication. The control application distributes messages to listening clients (visualisation platforms). Each client is listening on an open TCP socket, the first message sent to clients is NOP message. The connection remain opened until a CLOSE message is sent. The messages types and formats are described in following sections.

Message format

Each message is composed of four fields in this order: `MESSAGE_TYPE, OBJECT_ID, DATA_COUNT, data`. Fields are described in table 5.1.

message field	field type	description
MESSAGE_TYPE	string, 10chars	name and unique message identification, unused chars are spaces
OBJECT_ID	long64	object ID, message is related to
DATA_COUNT	long64	count of items in data field (*)
data	int32 or float32	

Table 5.1: Message fields and types.

(*) Data count is based on data types. If we are going to send one 3D point (6 floats, 1 int), data count will be 1. In case of binary data (e.g. image file) data count is Bytes count (see Table 5.3). All data types (long64, float32, int32) are coded in big-endian form.

Messages

Messages naturally corresponds to visualisation actions of 3D reconstruction editor. Messages should invoke actions described in table 5.4.

MESSAGE_TYPE	description	data
ADD_PC	add points to pointcloud	pc
UPD_PC	(update) delete all pointcloud points and insert new ones	pc
ADD_MESH	triangular mesh - add points	mesh
UPD_MESH	(update) triangular mesh - delete and insert mesh data	mesh
REMOVE	delete object (ID) - (DATA_COUNT=0)	no-data
VISIBLE	change object (ID) visibility to 0/1	byte
UPD_CAMERA	change position and orientation of camera (ID)	camera
NOP	no operation - for connection testing - (DATA_COUNT=0)	no-data
CLOSE	close connection - (DATA_COUNT=0)	no-data

Table 5.2: Messages used for reconstruction data transfer. Non-trivial data types are described in following section.

Structures

This section describes non-trivial data types used in synchronization protocol (data field).

structure name	data types	description
3D_point	3×float32, 3×float32, int32	position, normal, color(*) (DATA_COUNT=1)
pc	list of 3D_point	(DATA_COUNT=count(3D_point))
mesh	pc, long64, list of 3×int32	point data, count of triangular faces, indexes to point data (DATA_COUNT=count(pc))
camera	3×float32, 3×float32	position, euler orientation (DATA_COUNT=1)

Table 5.3: Message fields and types.

(*) RGB to int32 conversion:
`value = ((r & 0xFF) << 16) | ((g & 0xFF) << 8) | ((b & 0xFF) << 0)`

Actions

Actions corresponds to `MESSAGE_TYPE`.

action	description
`ADD_PC`	if not exist PC_{ID}, create new PC_{ID} `3D_point`s from message are added to PC_{ID} (existing PC_{ID} points are unchanged)
`UPD_PC`	if not exist PC_{ID}, create new PC_{ID} delete all PC_{ID} points `3D_point`s from message are added to PC_{ID} (in PC_{ID} are new points only)
`ADD_MESH`	if not exist $MESH_{ID}$, create new $MESH_{ID}$ data from message are added to $MESH_{ID}$
`UPD_MESH`	if not exist $MESH_{ID}$, create new $MESH_{ID}$ delete all $MESH_{ID}$ data data from message are added to $MESH_{ID}$ (in $MESH_{ID}$ are new data only)
`REMOVE`	delete any scene object OBJ_{ID} if exists
`VISIBLE`	change visibility of any scene object OBJ_{ID} if exists if `data==0` hide object, if `data==1` show object

Table 5.4: Actions detailed description. Abbreviations legend: PC_{ID} - poincloud ID (unique identification number), $MESH_{ID}$ geometrical object (mesh) ID, OBJ_{ID} - any scene object ID (geometrical, camera, image, etc.)

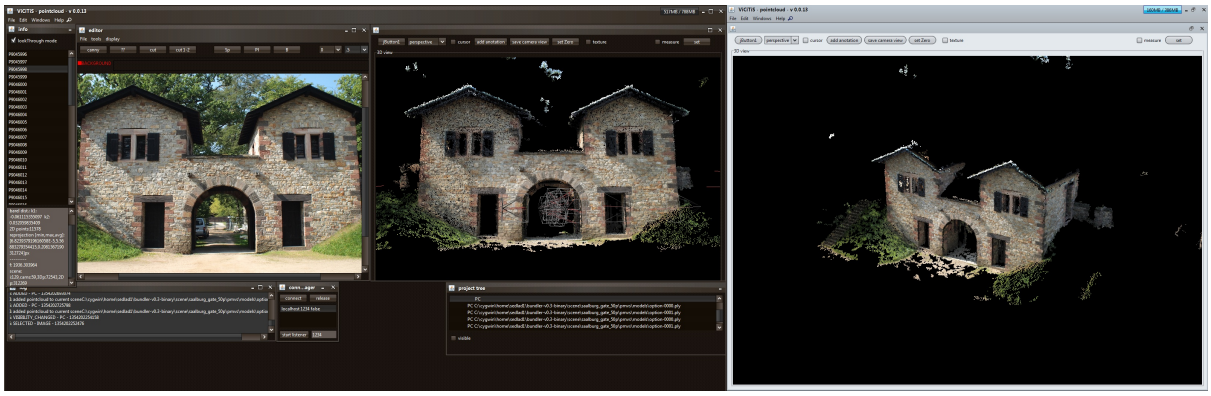


Figure 5.1: Synchronization between two instances of reconstruction software. Left is a master view with editing possibilities, on the right side is a visualisation client which can be on different computer and platform.

6 Conclusion and Future Work

That topic has proved important, prospective for further research, and with the potential for application in other disciplines (architecture, history, culture, etc.). At the same time we've detected the need for broader vocational point of view and cooperation in the context of the research methods of computer graphics (2D, 3D). Therefore, we plan to continue to address the issue within the framework of the newly served SGS application for the year 2013, in which we will deal with the topic of research in the field of photo-realistic imaging methods in real time.

In the course of the project we have established cooperation with experts from the Faculty of Civil Engineering, and together with them we are preparing an application-oriented project in which we want to apply the results of our SGS project (3D reconstruction of buildings). In cooperation with selected museums in the Czech Republic we want to transfer technologically undemanding methods into practical use in the field of heritage preservation. We are preparing the project for a grant program program NAKI of the Czech Ministry of culture.

Partial results of the SGS project were already published in [BTH+09], [BTH+10], [DKH09], [DZ11a], [DK12], [DSŽ11], [DZ11b], [SZ11], [SZ12], [STZ12], [UTZH12], [TB10].

Acknowledgements

This work has been funded by the Grant agency of the CTU Prague, grant No. SGS10/-291/OHK3/3T/13.

Bibliography

[BTH+09] R. Berka, Z. Trávníček, V. Havran, J. Bittner, J. Žára, P. Slavík, and J. Navrátil. Cave to cave: Communication in distributed virtual environment. Technical report, CESNET, z.s.p.o, 2009.

[BTH+10] R. Berka, Z. Trávníček, V. Havran, J. Bittner, J. Žára, P. Slavík, P. Borovský, and J. Navrátil. Implementing video-based, remotely accessible virtual environment system. Technical report, CESNET, z.s.p.o, 03 2010.

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, November 2001.

[CNSD+92] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The CAVE: Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM*, 35(6):65–72, June 1992.

[DC11] M. P. Deseilligny and I. Clery. Apero, an open source bundle adjusment software for automatic calibration and orientation of set of images. In *Proceedings of the ISPRS Symposium*, 3DARCH11 '11, 2011.

[DK12] J. Danihelka and L. Kencl. Interactive 3d services over windows azure. *Cloud Futures*, 2012.

[DKH09] J. Danihelka, L. Kencl, and R. Hak. Poster: Client-server talking head on mobile. In *International Conference on Advances in Computer Enterntainment Technology, Salon de ACE*. ACM, 2009.

[DSŽ11] E. Dušková, D. Sedláček, and J. Žára. Interactive Modeling and Visualization of Virtual Urban Spaces. In *Workshop 2011,CTU Student Grant Competition in 2010 (SGS 2010)*, pages 1–17, Praha, 2011. ČVTVS.

[DZ11a] J. Danihelka and J. Zara. Procedural generation of infinite cities. In *Eurographics 2011-Posters*, pages 31–33. The Eurographics Association, 2011.

[DZ11b] E. Duskova and J. Zara. Architectural Styles Dependent Shape Grammar Representation of Facades. pages 1–2, Llandudno, UK, 2011. Eurographics Association.

[EF07] P. Eisert and P. Fechteler. Remote rendering of computer games. In *SIGMAP'07*, pages 438–443, 2007.

[Ers] Ersi. CityEngine – 3D Modeling Software for Urban Environments. http://www.esri.com/software/cityengine.

[FP10] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010.

[GPSL03a] 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*. ACM, 2003.

[GPSL03b] 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*, GRAPHITE '03. ACM, 2003.

[GPSL03c] S. Greuter, J. Parker, N. Stewart, and G. Leach. Undiscovered worlds–towards a framework for real-time procedural world generation. In *Fifth International Digital Arts and Culture Conference, Melbourne, Australia*, 2003.

[HEB+01] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. Wiregl: a scalable graphics system for clusters. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, page 129140, New York, NY, USA, 2001. ACM.

[HKU+10] J. Halák, M. Krsek, S. Ubik, P. Žejdl, and F. Nevřela. Real-time long-distance transfer of uncompressed 4k video for remote collaboration. In *Future Generation Computer Systems*, 2010.

[JSH12] O. Jamriška, D. Sýkora, and A. Hornung. Cache-efficient graph cuts on structured grids. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[KBH06] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP '06, pages 61–70, 2006.

[LWWF03] 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.

[MWH+06] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. *Procedural modeling of buildings*, volume 25. ACM, 2006.

[PLH+90] P. Prusinkiewicz, A. Lindenmayer, J. Hanan, F. Fracchia, D. Fowler, M. de Boer, and L. Mercer. *The algorithmic beauty of plants*. Springer New York, 1990.

[PM01] Y. 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.

[SBZ12] D. Sedlacek, J. Burianek, and J. Zara. 3D Reconstruction Data Set - The Langweil model of Prague (Technical Report). Technical Report Series of DCGI CS-TR-DCGI-2012-3, Department of Computer Graphics and Interaction, Czech Technical University, FEE, november 2012.

[SSS06] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 835–846, New York, NY, USA, 2006. ACM.

[STZ12] D. Sedlacek, Z. Travnicek, and J. Zara. Multi-user Immersive 3D Reconstruction Environment. In A. Fusiello and M. Wimmer, editors, *EG 2012 - Posters*, pages 27–28, Cagliari, Sardinia, Italy, 2012. Eurographics Association.

[SZ11] D. Sedlacek and J. Zara. The Langweil Model of Prague - a Challenge for State-of-the-art 3D Reconstruction Techniques. In R. Laramee and I. S. Lim, editors, *EG 2011 - Posters*, pages 5–6, Llandudno, UK, 2011. Eurographics Association.

[SZ12] D. Sedlacek and J. Zara. User driven 3d reconstruction environment. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, C. Fowlkes, S. Wang, M.-H. Choi, S. Mantler, J. Schulze, D. Acevedo, K. Mueller, and M. Papka, editors, *Advances in Visual Computing*, volume 7431 of *Lecture Notes in Computer Science*, pages 104–114. Springer, 2012.

[TB10] Z. Travnicek and R. Berka. Multi-Threaded Real-Time Video Grabber. In *WSCG2010 Communication Papers Proceedings [CD-ROM]*, pages 259–263. Pilsen: University of West Bohemia, 2010.

[UTZH12] S. Ubik, Z. Travnicek, P. Zejdl, and J. Halak. Remote access to 3d models for research, engineering, and art. *IEEE Multimedia*, 19:12–19, 2012.

[VAB10] C. A. Vanegas, D. G. Aliaga, and B. Benes. Building reconstruction using manhattan-world grammars. In *CVPR*, pages 358–365, 2010.

[VABW09] C. Vanegas, D. Aliaga, B. Beneš, and P. Waddell. Interactive design of urban spaces using geometrical and behavioral modeling. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–10. Citeseer, 2009.

[Vam11] J. Vampola. Gramatika pro generování stávající pražské architektury. Bachellor's thesis, Czech Technical University in Prague, 2011.

[VAW+10] C. Vanegas, D. 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.

[Wim07] P. Wimmer. Anaglyph Methods Comparison, 2007. Available at `http://www.3dtv.at/knowhow/AnaglyphComparison_en.aspx`, last visited December 3, 2012.

[WMWG09] 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.

Appendix A

Overview of the main architectural styles in Europe

The appendix describes the architectural styles used mainly in Czech, Slovak and other European historical architecture. The appendix has been made by researcher Elena Dušková, which abandoned the project, however, we consider the results so important that we list them as part of this technical report.

Overview of the main ARCHITECTURAL STYLES in Europe

	ROMANESQUE	GOTHIC	RENNESAINCE	BAROQUE	CLASSICISM
Italy	1060-1250	1205-1420	**1420-1600**	**1570-1780**	1780-1830
France	**1000-1150**	**1135-1520**	1490-1610	1610-1770	1760-1830
Spain	**1000-1200**	1200-1510	1500-1600	1600-1800	1780-1830
Germany	1020-1250	1235-1520	1520-1660	1660-1780	**1755-1830**
England	1066-1200	1175-1550 Tudor style 1485-1550	Elizabethan era 1550-1610 Jacobean style 1610-1640	1670-1730 Georgian era 1710-1830	1820-1840

Sources:

- Wilfried Koch (Universum 2008): Encyklopedie evropské architektury od antiky po současnost
- José Pijoan (Tatran 1983): Dejiny umenia 3-4
- Carol Davidson Cragoe (2008): Abeceda architektury
- Gothic architecture and art: http://www.infoplease.com/ce6/ent/A0821376.html
- Visual Dictionary Online: http://visual.merriam-webster.com/

Pictures:

- Eugène E. Viollet-le-Duc (19th century): Dictionnaire raisonné architecture française XIe au XVIe siècle
  http://chateau.rochefort.free.fr/viollet-le-duc/
- Wikimedia Commons: http://commons.wikimedia.org
- my photo archive (2006-2011)

GOTHIC STYLE

Gothic style features according to importance (*Charakteristiky gotického štýlu podľa dôležitosti*)		
3 – the most important	**2**	**1**
pointed arch (*lomený oblúk*)	rose window – rosette (*rozeta*)	floral decorations (*rastlinné motívy*)
ribbed vault (*rebrová klenba*)	tracery (*kružby*)	niche / aedicule (*nika / edikula*)
flying buttress (*vonkajší oporný systém*)	finial (*fiála*)	mythical figures (*mýtické bytosti*)
high and narrow window (*úzke a vysoké okno*)	gargoyle (*chrlič*)	twin tower cathedrals (*dvojvežové katedrály*)
high and narrow building (*vysoká a úzka budova*)	steep roof (*strmá strecha*)	small side towers (*nárožné vežičky*)
	keystone (*svorník*)	diamond vault (*diamantová klenba*)
	compound pillar (*zväzkový pilier*)	baldachin (*baldachýn nad sochou*)
	console (*konzola*)	repetition of elements with small modifications (*opakovanie prvku s malými obmenami*)
	… (*profilácia*)	… (*cimburie tzv. lastovičí chvost*)
	shouldered arch (*sedlový oblúk*)	
	ogee arch (*oslí chrbát*)	
non geometrical (*negeometrické*): stained glass (*vitráž*) lightness (*svetlosť a ľahkosť priestoru*)		

flying buttress:
Masonry structure in the shape of a partial arch; it supports a wall by transferring the pressure of the vaults onto an abutment.

rose window:
Large circular bay composed of decorative tracery and stained glass; it is also called a rosette.

keystone (*svorník*):
Found on the top of the vaults Nachádza sa vo vrchole rebrovej klenby.

console (*konzola*):
Hlavica, ktorá je pripevnená k stene a nemá základový stĺp. Slúži ako podporný prvok strechy, klenby, oblúkov alebo sôch. Býva zapustená do nosnej steny a sama drží napr. balkón alebo sochu či klenbové rebrá, môže na ňu nadväzovať pilaster (stĺp „prilepený" k stene).

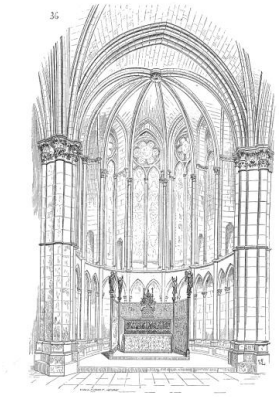
niche (*nika*)
Vyhĺbený výklenok v stene, určený na sochu.

aedicule (*edikula*)
Malá otvorená stavba na umiestnenie sochy, zadnou stranou priliehajúca k stene.

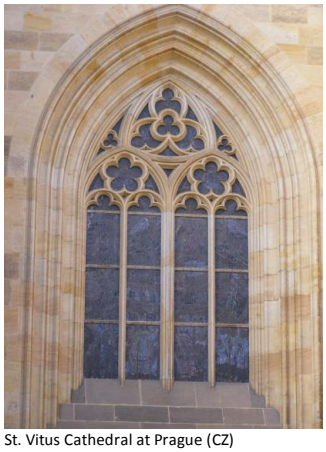
stained glass:
Translucent decorative work comprised of an assemblage of glass pieces, usually colored, that fills a bay.

2

3 - pointed arch (*lomený oblúk*)



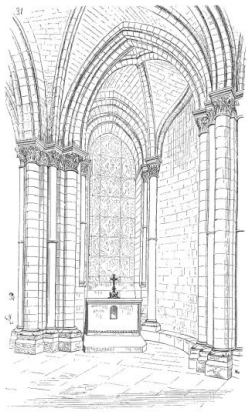
Cathedral Reims (F)



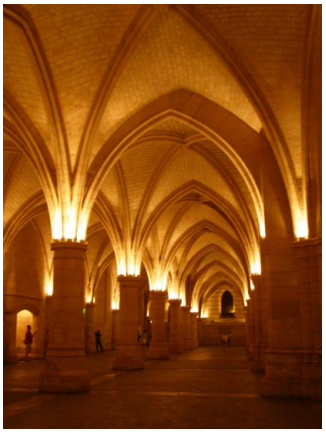
St. Vitus Cathedral at Prague (CZ)

St. Vitus Cathedral at Prague (CZ)

3 - ribbed vault (*rebrová klenba*)



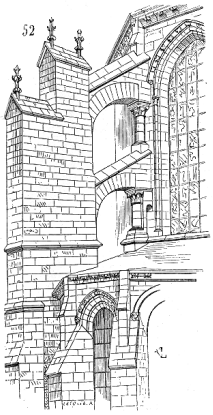
Illustrative example



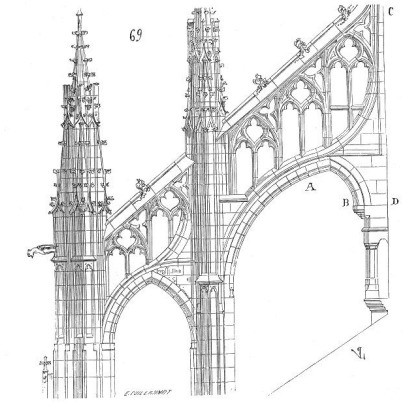
Conciergerie Paris (F)

Old Town Square Prague (CZ)

3 - flying buttress (*vonkajší oporný systém*)



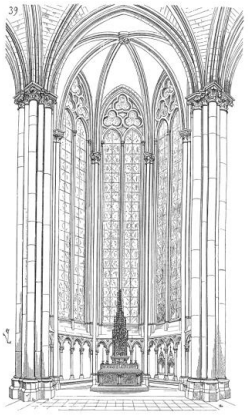
Cathedral Soissons (F)



Saint Wulfran Church at Abbeville (F)

St. Vitus Cathedral at Prague (CZ)

3 - high and narrow window (*úzke a vysoké okno*)

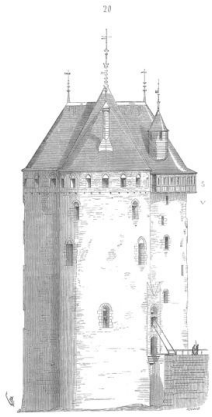


Cathedral Amiens (F)

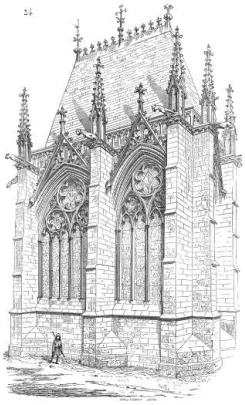
St. Vitus Cathedral at Prague (CZ)

St. Elisabeth Cathedral at Košice (SK)

3 - high and narrow building (*vysoká a úzka budova*)



Donjon Etampes (F)

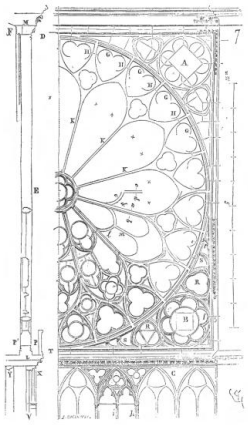


Cathedral Mantes (F)

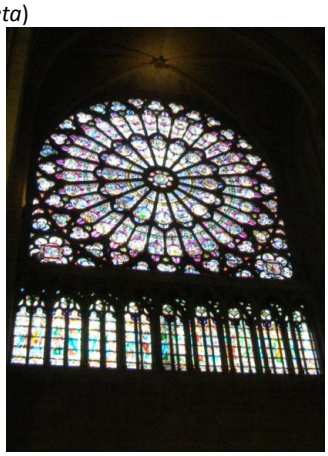
Powder Tower Prague (CZ)

---

2 - rose window – rosette (*rozeta*)



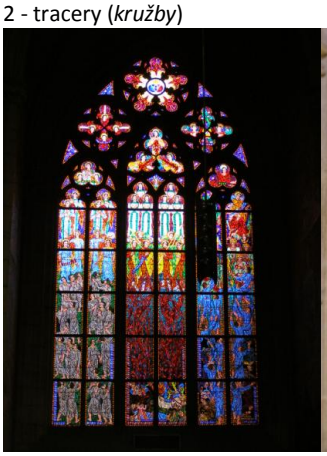
Notre Dame Paris (F)

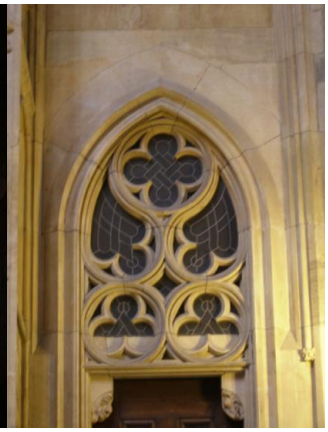


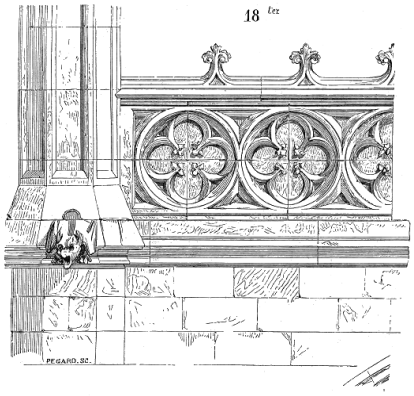
Notre Dame Paris (F)

St. Vitus Cathedral at Prague (CZ)

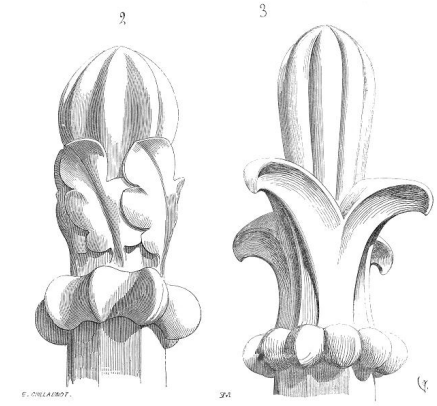
2 - tracery (*kružby*)


St. Vitus Cathedral at Prague (CZ)


St. Vitus Cathedral at Prague (CZ)


Cathedral Beziers (F)

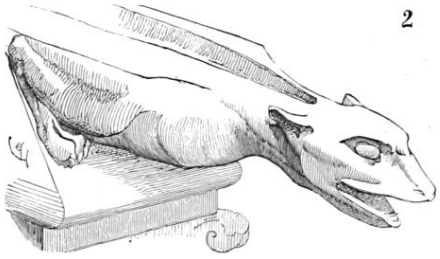
2 - finial (*fiála*)


Cathedral Paris (F)

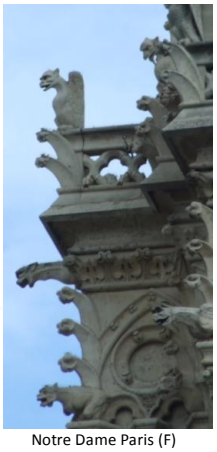

Church Poissy (F)


St. Elisabeth Cathedral at Košice (SK)

2 - gargoyle (*chrlič*)

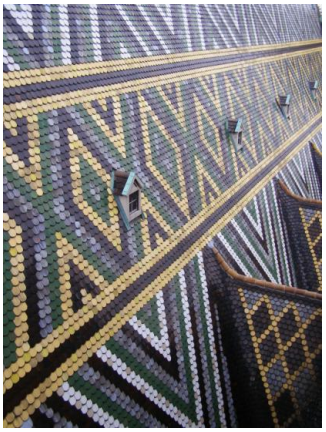

Illustrative example


Illustrative example


Notre Dame Paris (F)

2 - steep roof (*strmá strecha*)

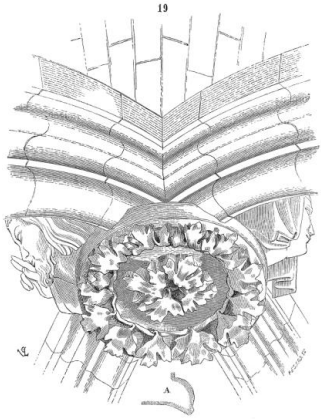
Church Ry (F)



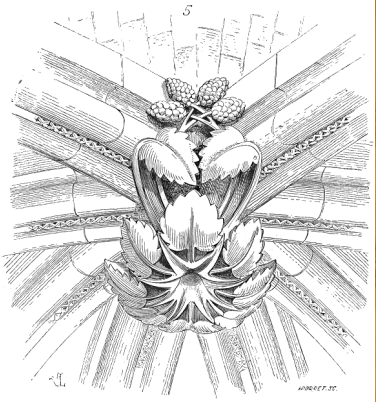
St. Stephen's Cathedral, Vienna (A)

St. Michael Chapel at Košice (SK)

2 - keystone (*svorník*)



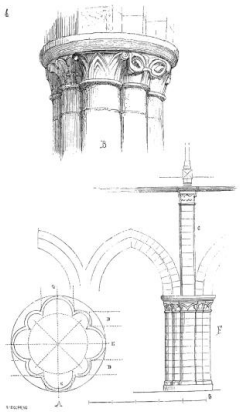
Cathedral Carcassonne (F)



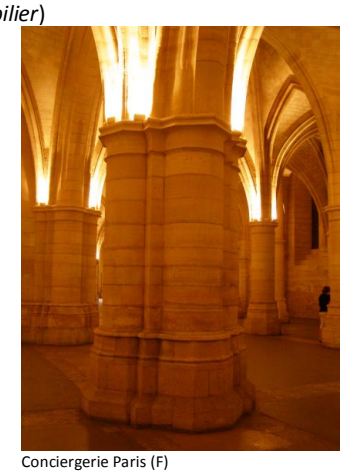
Cathedral Laon (F)

Powder Tower, Prague (CZ)

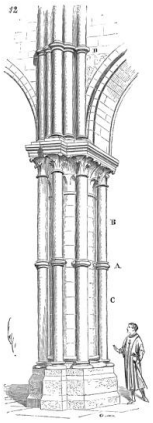
2 - compound pillar (*zäzkový pilier*)



Church Guerande (F)

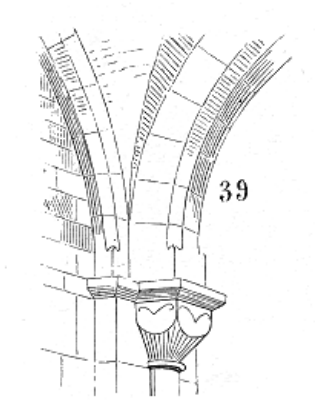


Conciergerie Paris (F)



"en délit" Notre Dame Laon (F)

2 - console (*konzola*)

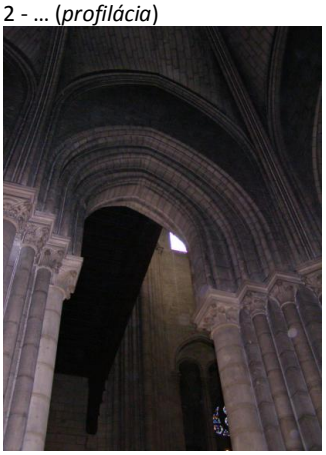


Illustrative example


Old Town Hall, Prague (CZ)


Powder Tower, Prague (CZ)

2 - … (*profilácia*)


Notre Dame Paris (F)

2 - shouldered arch (*sedlový oblúk*)


The Residence of the Chamber Count, Kremnica (SK)


St. Elisabeth Cathedral at Košice (SK)

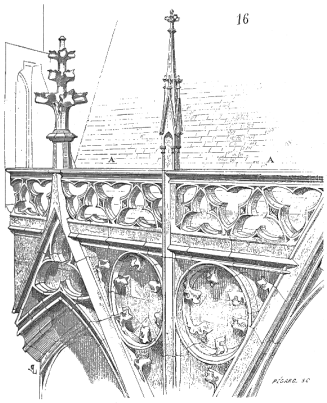

Mediaeval house, Kremnica (SK)

2 - ogee arch (*oslí chrbát*)

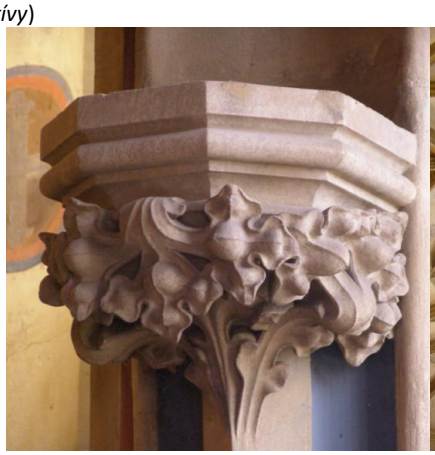
Mediaeval house, Kremnica (SK)

---

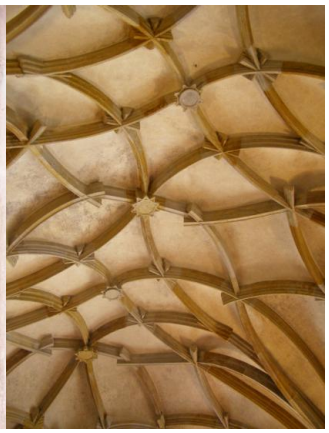
1 - floral decorations (*rastlinné motívy*)



Church Saint Urbain Troyes (F)

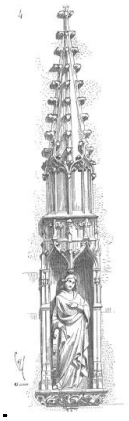


Old Town Hall Prague (CZ)



Prague Castle (CZ)

1 - niche / aedicule (*nika / edikula*)

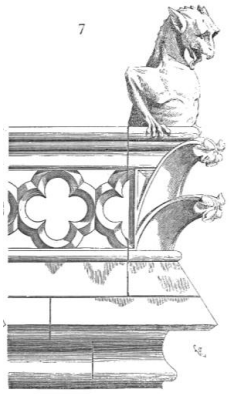


Niche fifteenth century (F)

Notre Dame Paris (F)

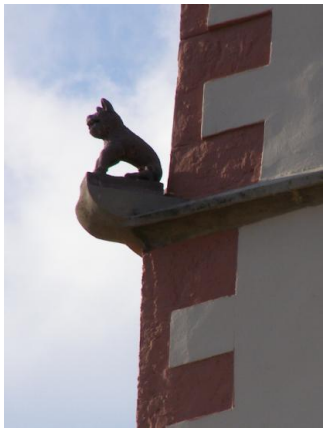
Notre Dame Paris (F)

1 - mythical figures (*mýtické bytosti*)



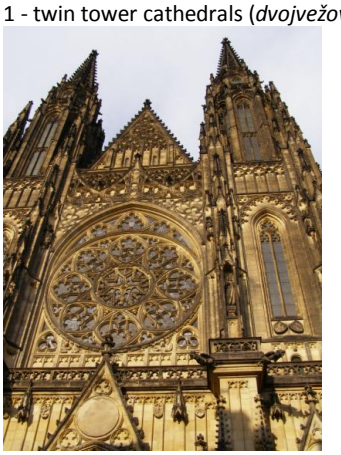
Cathedral Paris (F)

St. Elisabeth Cathedral at Košice (SK)



Church of St. Catharine at Kremnica (SK)

1 - twin tower cathedrals (*dvojvežové katedrály*)

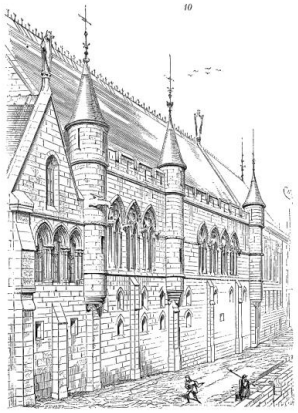


St. Vitus Cathedral at Prague (CZ)

Westminster Abbey (UK)

Cathedral Reims (F)

1 - small side towers (*nárožné vežičky*)

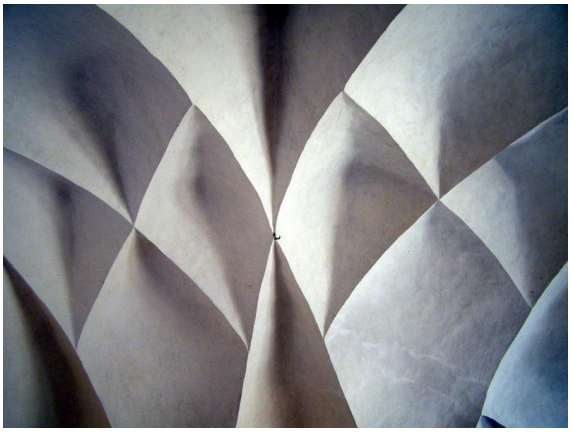


Illustrative example

Powder Tower, Prague (CZ)

1 - diamond vault (*dimantová klenba*)

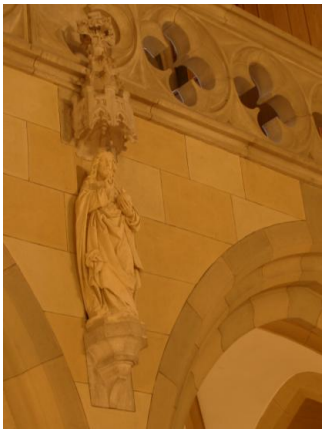
Mediaeval house, Kremnica (SK)



The Residence of the Chamber Count, Kremnica (SK)

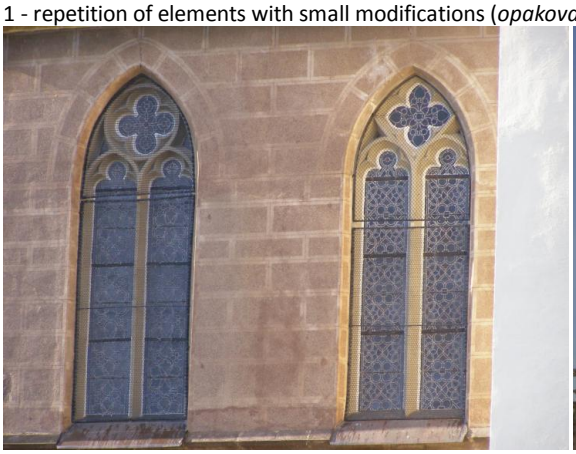
1 - baldachin (*baldachýn nad sochou*)

Church of St. Catharine at Kremnica (SK)

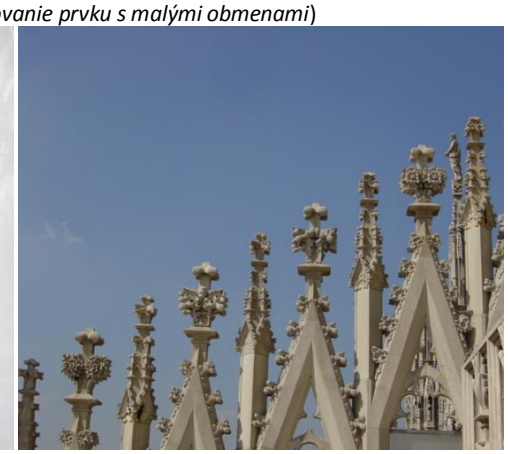


Church of St. Catharine at Kremnica (SK)

1 - repetition of elements with small modifications (*opakovanie prvku s malými obmenami*)



Church of St. Catharine at Kremnica (SK)



Cathedral Milan (I)

1 - … (*cimburie tzv. lastovičí chvost*)

Prague Castle (CZ)

11

Typical buildings:

Cathedrals

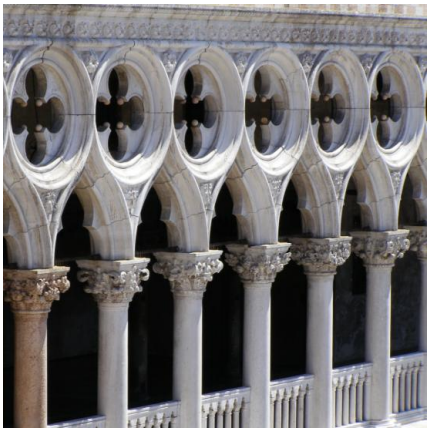
- Chartres Cathedral (F)
- Reims Cathedral (F)
- Notre Dame de Paris (F)
- St. Stephen's Cathedral, Vienna (A)
- Westminster Abbey (UK)
- Milan Cathedral (I)

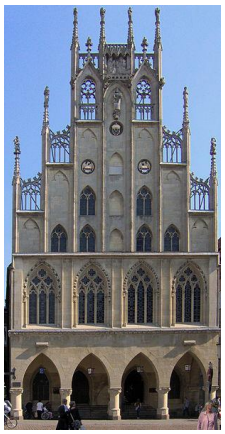
Religious buildings

- The Popes' Palace, Avignon (F)
- Monastery, Santes Creus (E)

Secular buildings

- Doge's Palace, Venice (I)
- Conciergerie, Paris (F)
- Town Hall, Münster (D)
- Silk Exchange, Valencia (E)
- Castel del Monte (I)



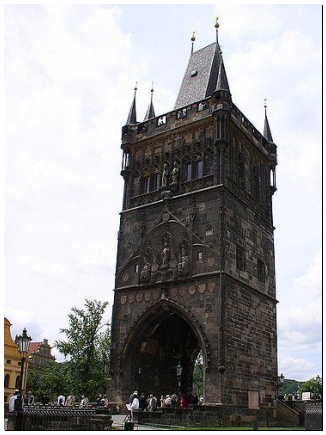


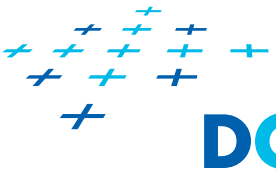
Gothic in Czech Republic:

- St. Vitus Cathedral, Prague
- Church of Our Lady before Tyn, Prague
- Stone Bell House, Prague
- Old Town Bridge Tower, Prague
- St. Barbara Cathedral, Kutná Hora

Gothic in Slovakia:

- St. Elisabeth Cathedral, Košice





Technical Report Series of DCGI, Volume 2, Year 2012

Department of Computer Graphics and Interaction
Czech Technical University in Prague, CZ
Faculty of Electrical Engineering
Website: http://dcgi.fel.cvut.cz

Karlovo nám. 13
121 35 Praha 2
Czech Republic

Tel: (+420) 2 2435 7557
Fax: (+420) 2 2435 7556