City University London

MSc in Advanced Computer Science

Project Report

2015

Adding a Dimension:
Turning a Famous Painting into a 3D Scene

Adriána Daniláková

Supervised by: Dr. Gregory G. Slabaugh

25/09/2015

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: *Adriána Daniláková*

Abstract

A single view geometry reconstruction is still an open problem in the computer vision field. Certain advances have been made in creating 3D models from a single source image given prior knowledge of the scene and objects. In this project, we present a sophisticated approach to reconstruction and completion of 3D scene from an image of a painting of semi-structured open environment. The absence of an unequivocal solution for the reconstruction of non-photorealistic images is compensated for by the application of existing theories and expertise from various areas of human visual perception as well as precepts of geometrical art analysis in order to produce a plausible 3D representation of the image content for a human audience. Our method involves a user working with existing graphics software and tailored computer vision algorithms in a process flow of dissection and analysis of a source painting, processing its components, reconstruction of individual parts, and synthesizing the final scene. We demonstrate the feasibility of our method by producing a virtual environment walkthrough and online environment documenting our techniques and artefacts.

Keywords

2D to 3D conversion, single view reconstruction, scene completion, visual perception bias, science of art

Acknowledgements

First, I would like to thank my supervisor Dr. Greg Slabaugh for his valuable insights and guidance for the duration of this project and for introducing me to the magnificent world of computer graphics and vision. I wish to acknowledge all my City University lecturers who have contributed to my current level of knowledge and who made my year of study a very interesting experience.

Further, I would like express my appreciation to people providing free online study materials and courses that helped me to understand many topics. My grateful thanks also extend to people who volunteered to participate on my project.

Finally, I wish to express a gratitude to my family and friends for their endless support and encouragement.

Table of Contents

1. Introduction and Objectives

3D modelling is widely used across different industries including movies, animation and gaming. Techniques applied in movie and game modelling are similar, however there is a big difference in procedures and model quality limitations given by real-time rendering required in computer games and simulations (Masters, 2014).

The last decade of evolution in computer graphics however brought significant advances. Different standard approaches to rendering enable a developer to create various degrees of realism of 3D models (Slabaugh, 2015). OpenGL became one of the most popular APIs for 3D graphics thanks to, among the other reasons, shader programs, giving more flexibility and power to the hands of developer (Wolff, 2011, p. 7). Together with ever-increasing computational performance, these newer methods enable creation of high quality graphics environments for games and simulators.

The computer graphics field is sometimes characterised as a forward problem – with given scene geometry, light reflectance, and atmospheric conditions, a goal is to synthesize the final image (Malik, 2012). An inverse problem to this relates to computer vision – trying to recover information about the world from one or more images to reconstruct its properties, such as position, shapes, and illumination (Szeliski, 2010, p. 5).

Researchers studying computer vision developed procedures that produce accurate 3D models from several photographs that individually capture objects from different angles. Moreover, algorithms have been designed to compute the 3D position of the points that create a match only between two images from different camera angles (Hartley and Zisserman, 2004).

However, the problem of scene reconstruction from a single image remains open. With several assumptions, a closed space of the room (Hartley and Zisserman, 2004, p. 10), or a very symmetrical building of Taj Mahal (Malik, 2012) have been modelled from single image. Algorithms for creating depth model of the scene have been published with partially promising results (Saxena *et al.*, 2007) and also methods for various single objects modelling (Kholgade *et al.*, 2014).

The reason why solving a problem of reconstruction of 3D scene from a single image would be useful is among others, the sheer volume of various image sources available (in contrast with the multiple images of the same scene that must have certain qualities in order to be used for reconstruction). Many budding game developers are interested in developing game functionality, but don't have artistic talent or skills to create pleasing visual environments for their games and hiring a professional graphics artist is not an option. Being able to choose from many 2D art images and convert it to 3D environment would certainly create a benefit.

Reconstructed 3D scenes from art images would be certainly appealing to the groups other than game developers. How many times one would like to enter the painting and see its world from inside? At the end, this project would enable that kind of experience.

1.1. Research Question

Based on gaining experience with OpenGL in the course of Computer Graphics module and having a lifelong art appreciation, the following research question was formulated:

Is it possible to perform a conversion of a painting into a fully contained 3D graphics scene as one person's job, using commonly available software and tools without losing the artistic experience?

With all publicly available research and tools, and relatively high performance hardware in average computers it seems like a reasonable time to add one more dimension and take artistic experience to new level.

1.2. Aim and Objectives

The aim of this project is to provide the proof of concept of a production method that enables a skilled human operator, using specialized tools and techniques, to create a computer graphics application, which will simulate the walkthrough of a 3D scene, visually based on a chosen painting.

To accomplish this aim, several project objectives have been defined as follows:

Objective	Testable result
Identify and evaluate existing tools and software that can be used in the conversion process.	List of tools and software with the pros and cons of their usage.
Identify criteria for painting selection.	Specified recommendations for painting selection/rejection.
Design and develop visually pleasing and accurate individual components of the scene.	Skybox, terrain, meshes based on 2D representation from painting.
Create plausible complementary parts of the scene to those in the painting.	Complete seamless 3D scene.
Design interesting track around the scene and versatile camera modes to present the scene.	Different modes of camera view regulated by users with possibility of virtual tour.
Perform evaluation of final scene by wider group.	Evaluation of the scene via questionnaire.
Contribute to "world's body of knowledge" (Dawson, 2009, p. 17).	Learning material documenting the production method that would allow future users to try it.

Table 1: Project Objectives

1.3. Work Products

At the end of the project, it is intended to deliver the following products:

- **Computer graphics application** – designed for Windows platform using OpenGL API together with OpenGL Shading Language and other related C++ libraries. The application will provide a virtual tour around the painting environment, enabling users to interact with it – changing the speed of the tour and setting different camera views. The painting environment should be true to the original and there should be no obvious boundary between the parts shown on painting and the rest of the scene.
- **Conversion User Guide** – that will document conversion process in order to provide information for future users who would like to repeat the process. It will be designed as modern learning material with links to used tools and resources, and it will also contain information about alternative tools and techniques that were examined during the research process but weren't used at the end.

1.4. Project Beneficiaries

This project expects to produce useful outputs for following groups of potential beneficiaries:

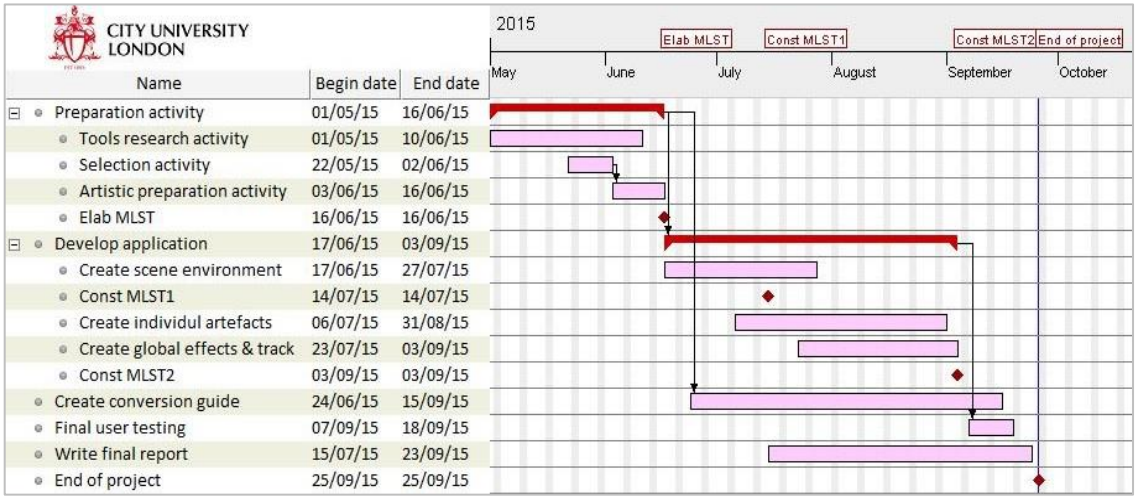
- **Game developers** – the Conversion User Guide can help game developers who lack ability in graphics design and want to focus on game functionality to create scenes for their game using 2D image sources. Individual elements of the application design (skybox, textures, and meshes) will also be made publicly available so they can be reused in other computer graphics projects.
- **Computer graphics students** – the Conversion User Guide will serve students who want to explore modern computer graphics features beyond their basic computer graphics course. City University London's students in particular can benefit from the additional information from the Guide since it will build upon topics taught in the computer graphics module. Source code of the application can also be provided.
- **Art educational institutions and galleries** – the application can be offered to these organizations for education or entertainment. It will be easily possible to add narrative about the painting which can guide a virtual tour around it.
- **Researcher** – as well as from outputs of project that can be included into work portfolio, the author as researcher will benefit from the whole research experience by gaining insight into academic research procedures, learning how to create 3D meshes, improving OpenGL knowledge and keeping C++ programming skills fresh.

1.5. Methods Outline and Project Plan

For the computer graphics application development it is necessary to employ a structured and well defined software development methodology. Because of relatively high uncertainty and technological nature of this project, iterative and incremental method was decided as the most suitable. Representing this type of method, RUP (**R**ational **U**nified **P**rocess) provides a customizable framework for a specific project, defining phases of the project and process disciplines (Shuja and Krebs, 2008). An artistic nature of the project (Oates, 2009, p. 118-119) had an influence on the whole development process.

For the Conversion User Guide in online form a less structured approach is more suitable. Rapid application development enables the creation of core functionality early and to adjust solution to newly recognised requirements in the course of the project.

The main project stages were planned around the RUP phases; with project Inception phase finishing with the project proposal approval. The four stages of project with Elaboration, Construction and Transition milestones are shown in the following Project Plan. More detailed information about project planning is available in Project Proposal in Appendix A.



Name	Begin date	End date
Preparation activity	01/05/15	16/06/15
Tools research activity	01/05/15	10/06/15
Selection activity	22/05/15	02/06/15
Artistic preparation activity	03/06/15	16/06/15
Elab MLST	16/06/15	16/06/15
Develop application	17/06/15	03/09/15
Create scene environment	17/06/15	27/07/15
Const MLST1	14/07/15	14/07/15
Create individul artefacts	06/07/15	31/08/15
Create global effects & track	23/07/15	03/09/15
Const MLST2	03/09/15	03/09/15
Create conversion guide	24/06/15	15/09/15
Final user testing	07/09/15	18/09/15
Write final report	15/07/15	23/09/15
End of project	25/09/15	25/09/15

Figure 1: Project Plan

1.6. Changes during Project

The scope of changes that occurred in the course of the project is limited. There were some minor changes in the timing of terrain and meshes design and also in global effects incorporation. A partial testing by small group of users (Project proposal) during development turned out to be unnecessary, because individual object design was tested objectively via overlaying over originals and only once the whole scene was assembled user feedback was necessary (provided by final user testing). There were few refinements concerning Conversion User Guide – it turned out in the early stage of the project that an online version would be best for our purposes. Other changes included tools used for the work: local versioning of the code turned out to be sufficient, the OpenCV library was used instead of the Boost library and Photoshop tool was included as common tool due to cheap licensing options available.

1.7. Project Report Outline

The remainder of the project report is organised as follows:

- Chapter 2 – **Context** provides theoretical background for the project's research process. Explanation in this chapter covers crucial notions of human perception, geometric projections, science of linear perspective in the art creation and computer vision algorithms relevant to our research. Subsequently, an OpenGL API potential is demonstrated in relation to the project products. The following section describes the existing research work in 2D to 3D conversion and partially relevant computer vision solutions. Finally, a short section is devoted to good practice in online presence and learning.
- Chapter 3 – **Methods** presents the sequence of techniques employed in the different stages of the project life-cycle. The main principle, an iterative and incremental approach to development is applied in various ways to each project product. Relatively rigorous methods of software development are used in combination with more unstructured design processes. At the end, both technical and user evaluation is performed for partial products as well as the final project product.
- Chapter 4 – **Results** encompasses presentation of the project products and their breakdown into individual components. Besides final versions of the products, prototypes are shown to illustrate an evolution of the products as well as intermediate artefacts not used as a part of final solution but necessary for their production. The chapter contains separated sections for computer graphics application and for Conversion User Guide.
- Chapter 5 – **Discussion** review project results in relation to the originally defined objectives and also in the wider context of other research done in the area. Various aspects of project approaches are discussed such as their robustness and future applicability. The limitations of the automation of the whole process are discussed too.
- Chapter 6 – **Evaluation, Reflections and Conclusions** reviews the original project proposal in the light of final accomplishments. Besides the tangible project products, professional growth as a result is considered as well. It finishes with the suggestions for possible results applications and future research in related computer vision area.

2. Context

The research challenge presented by this project spans two distinct areas of computer science: computer vision and computer graphics. The production pipeline starts with the application of computer vision algorithms and procedures in order to recover information for 3D scene modelling, which is then assembled and synthesized into a virtual environment with the help of computer graphics tools and techniques. This process can be anywhere on scale from rigid, following certain guidelines and techniques to free artistic production. The ambition of this project is to find a guideline for anyone regardless of artistic skills.

So what is the current state of computer graphics and computer vision in relation to the project needs? As mentioned in the first chapter, computer graphics deals with a 'forward' problem and even though it is not yet capable of photorealistic rendering in real-time (Andrews, 2013) there are techniques to enhance photorealism of models without sacrificing the performance (e.g.: texture baking from high-poly models applied on low-poly models). For the project using a non-photorealistic painting as a source, this is not a very serious issue, even though with numerous objects in a scene performance remains constant concern.

Even though computer vision has a similarly long history as computer graphics, an inverse problem has still not been satisfactorily resolved – that of incomplete information retrieval from the source that is needed for full scene reconstruction (Szeliski, 2010). According to Ma, it is not possible to "invert the image formation process and reconstruct the true scene from a number of images" (Ma *et. al.*, 2005, p. 2). Therefore computer vision scientists are looking at the laws of physics, optics and statistics as a guide to find correct solutions or at least find interpretations of an image that are more likely to be valid (Malik, 2012).

A suggestive source of inspiration for computer vision scientists of how to cope with visual uncertainties of the surrounding world would be human vision and perception. Even though the aim of computer vision is not to reproduce a human vision, many times it is a human user who judges the results of computer vision algorithms (Malik, 2012). There are also objective physical measurements that can be applied to verify results; however it is not always possible to apply them, as happens to be the case for a painting. For that reason it is important to understand human perception and apply this knowledge in conversion process.

Another clue for scene reconstruction can be gained from understanding the science that accompanied the development of painting since the Renaissance. A mathematical system of linear perspective was developed in order to create a consistent illusion of 3D space and guidelines used by artists can be observed in final paintings. When we combine all these observations with projective geometry and apply computer vision techniques, we should be able to create a 3D scene that would seem plausible to human audience considering an original 2D source.

Then we need to know about the abilities and limitations of OpenGL to be able to synthesize the scene for the application. Knowledge of existing research provides us with useful inspiration and identifies gaps in current 3D reconstruction landscape that haven't been addressed yet. At the end of this chapter, we discuss approaches to software development and good practice in online presence for the Conversion User Guide.

2.1. Information Acquisition and Recovery

2.1.1. Human Vision and Perception

Our perception of the world is limited to our sensory inputs[1]. For the majority of people, sight is considered to be their most important senses and main information source about the surrounding environment (Politzer, 2008). For an illustration of visual perception functioning, many times a simplistic depiction is used, describing it as light inputs activating photoreceptors on the retina and then an image being transmitted and displayed on a screen in our brain. However, our visual perception involves much more than just creating a replica of reality in the same way as it is captured by a camera. Indeed, judgment of the brain is an integral part of every act of perception (Ramachandran, 2005, p. 67).

The perpetual interaction with the brain makes sense when we consider visual perception from the evolutionary point of view – the whole point of the evolutionary development of sight was to allow an organism to see things that in turn generate successful survival behaviour of the organism. That means that the brain has developed several levels of assumptions and filters, which sometimes cause deviation of what humans perceive from objective physical measurements of the world, but in most cases (not always, see Figure 2) we can interpret what we see sufficiently well (Purves, 2010).

Figure 2: Making assumptions

3D drawing of the glass of water, 3D street art, and picture of kissing Sphinx – all these images can illustrate brain filtering and interpretations. Our first reactions might be to grab the glass of water, avoid the giant hole in ground and think the woman and the statue are the same size. After a while we realise that big hole doesn't correspond with the rest of the environment in the image and that statue is well known Egyptian Sphinx and we prioritise certain visual cues more than the others. However it is hard to infer more about the glass of water just from the single image. (Photo credits: from left Deviant Art - Sany Lebedev, Metanamorph - © Edgar Mueller, Flickr - The Black Azar)

According to Purves, at present scientists cannot explain human perception purely via standard anatomical and physiological approaches to vision and he suggests to consider the phenomenology of what we see instead (Purves, 2015). However, besides the phenomena of human perception there are a few physiological characteristics of human visual system that can indicate a lot about human perception.

[1] Besides five traditionally recognised senses (sight, hearing, taste smell and touch) there are also sense of temperature, balance, pain, vibration and kinaesthetic ('Sense', 2015).

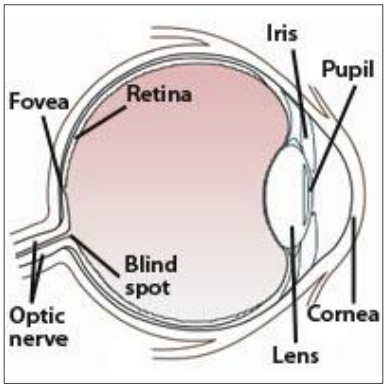


Figure 3: Human eye

The image formation process starts in the eye (Figure 3) with light rays hitting the cornea where they refract and continue further towards the iris, which controls the diameter of the pupil and thus regulates the amount of light reaching the retina. Behind the iris is the lens that focuses light onto the retina, whether for near or distance vision. The retina is the place where all photosensitive receptors are. There are two types of photoreceptors: rods, which are light sensitive but don't provide a sharp image, and cones, which are colour sensitive and give a sharp image. A special part of the retina is called the fovea and is the location with the highest concentration of the cones. The optic nerve carries signals from rods and cones to the brain; at the place where it connects to the eye is the so-called 'blind spot', an area completely lacking rods and cones. (Cohen and Wood, 2000, p. 186).

When we examine the density and distribution of photoreceptors (Figure 4), there are some interesting aspects to consider. The concentration of the cones at the fovea region is around 147,000 per square millimetre (Shroff, 2011, p.97), with the fovea region being about 1.5 mm wide, the total number of photoreceptors is approximately 350,000. If we compare this number for example with camera resolution, based on quality of our image of the world it is clear that vision doesn't work as holistic image on pixel by pixel basis but instead involves some form of visual information compression (Radke, 2015).

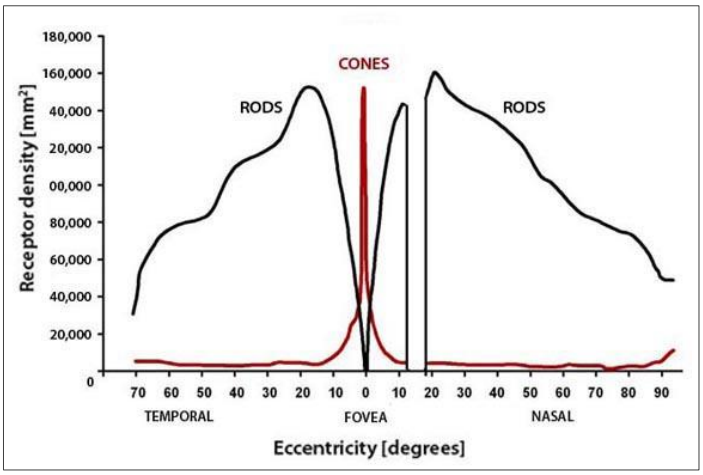


Figure 4: Concentration of photoreceptors on the retina (based on Osterberg's measurement)

Solso suggests that information from photoreceptors is hierarchically processed firstly by the visual cortex, searching for edges, lines and contrasts (salient features in general) that are passed to cerebral cortex for higher order processing and analysis, and object knowledge at the end, which initiates movement of the eyes to look for further details and obtain new images (Solso, 1996, p. 31). Similar processing is also described by Ramachandran, with visual inputs going via two pathways with each stream being processed in specialised visual areas of the brain (Ramachandran, 2005). If this is the case, correct transformation of the defined lines and prominent shapes from a painting into a 3D scene should be our priority to make it recognisable to human audience.

The next interesting feature of rods and cones arrangement is the blind spot area. There is no sensory input at this part of retina. Nevertheless we have no void in our visual image. One possible explanation would be for brain compensating with an image from the other eye to gain missing visual information, but there is no void when we have only single eye open either. The phenomenon of dealing with gaps in the visual image is called filling-in and it is even more pronounced in case of scotomas – parts of the vision field with diminished or completely degenerated visual acuity. This filling-in can reproduce colours, textures, repetitive patterns but not face or complex objects (Sacks, 2010, p. 176). Ramachandran calls it perceptual filling-in (there is also conceptual filing-in in form of hallucinations) and argues that it is not a high-level cognitive process. But he also recognises other fillings-in that happen at different stages of the visual process, e.g. filling-in of the shape of an animal that we see behind a fence (Ramachandran, 2005). Filling-in is interesting in our research because it provides a model for how we can obtain missing image parts created during segmentation of the painting due to occlusion of one object by the other.

Now we consider the phenomena of human perception, which can be potentially useful in solving our computer vision problem. Purves states that our perception is misleading with the respect to reality as a result of our visual system trying to deal with inverse problem of vision. This inverse problem applies both to luminance and geometry.

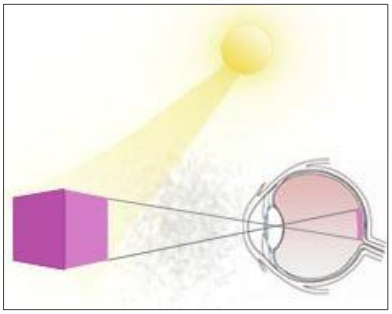


Figure 5: Inverse problem – luminance (based on Purves' image)

The problem with perception of luminance (Figure 5) means the inability to separate the effects of scene illumination sources, surface reflectance of the objects in the scene, and the atmosphere influence on transmission of the final image to the retina. This has alteration effects on perception of colours, but also on appraisal of surface curvature of the observed objects. If human vision wasn't able to deal with this problem at all, we would see objects in different illuminations as different sorts of things (Purves, 2015).

The inverse problem of vision in relation to seeing geometry (Figure 6) was first described almost three hundred years ago. Its basic premise is that we can't definitely decide a distance, size and orientation of the 3D object in the world based on 2D image projected onto the retina (Purves, 2010, p. 120). But because we live in the world with stable physical properties, our brain integrates into visual processing certain assumptions, gained partially via evolution and partially through the learning process in childhood, in order to remove ambiguities in perception (Ramachandran, 2005, p. 68).

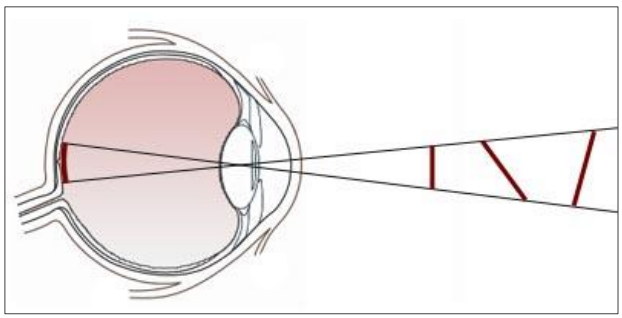


Figure 6: Inverse problem – geometry (based on Purves' image)

When we look at the problem of identifying position of still objects in the world, the human visual system has several different methods for its assessment. Having two eyes, each with slightly different image of the world, a human has the perception of depth and three-dimensionality called stereopsis. Yet, there is around 5 to 10 of population lacking stereopsis partially or completely, who can navigate around the world perfectly well (Sacks, 2010, p. 120). These people can judge distance of the objects via combination of several monocular (pictorial) cues (Figure 7).

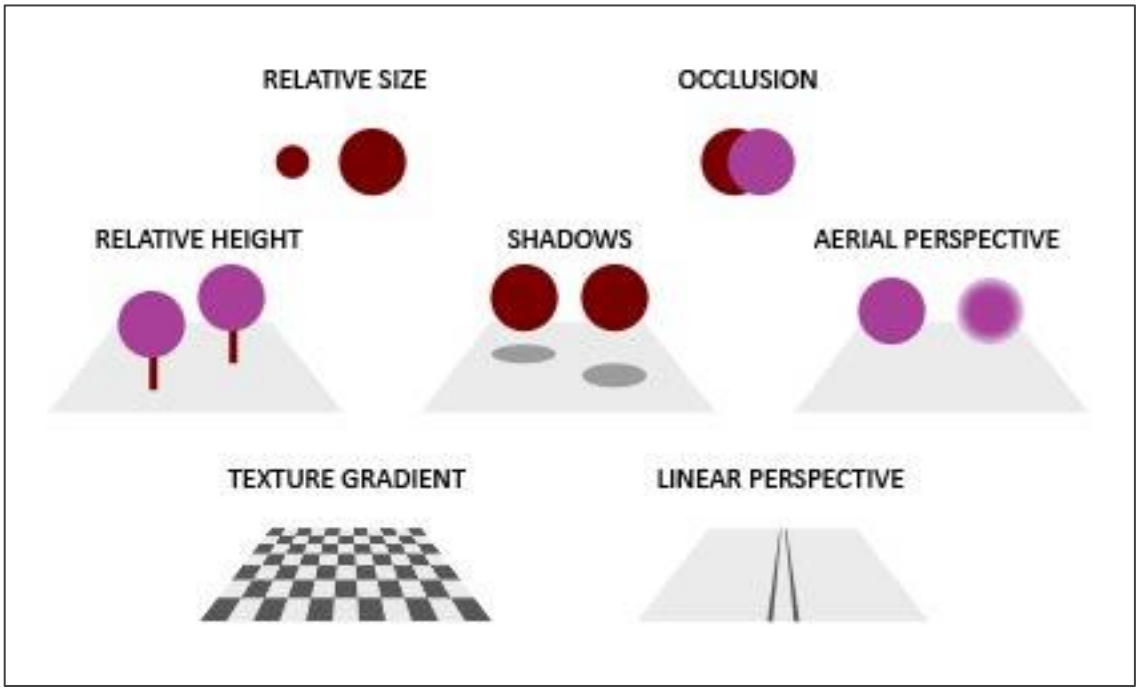


Figure 7: Monocular cues relevant for assessing distance from still image

Interpretations of monocular cues are the following (Sedgewick, 2004)(Krantz, 2010):

- **Relative size –** size of an object is larger, when object is closer to the observer.
- **Occlusion –** an object that is obscuring the other object is considered to be closer to observer.
- **Relative height –** when object is on ground lower than the other object, it is considered to be closer. When object is in the air (over horizon), the closer object is higher than the more distant one.
- **Shadows –** with most illumination coming downwards, we can infer object distance from position and shape of shadow on ground, with closer object having shadows lower.
- **Aerial perspective –** more distant objects are hazy and bluish (caused by atmospheric particles between observer and distant object).
- **Texture gradient –** closer textures are clearer and more defined, more distant textures are more indefinite.
- **Linear perspective –** parallel lines seems to converge in distance (on horizon).

Once the distance of an object is established, there is one unknown parameter less in the distance-size-orientation relation. Knowing just this parameter, the height can be easily computed (more about that in Section 2.1.2). That works also the other way around – for objects of known heights, a distance from the projection plane can be computed quite precisely. When assessing distances based on objects' known size it is always better to choose ones with small size variations (e.g.: human figure) than one for which sizes can vary in large measurement units (e.g.: trees). When assessing orientation of objects in relation to the surface of ground, considering gravitational force effects can also provide valid cues.

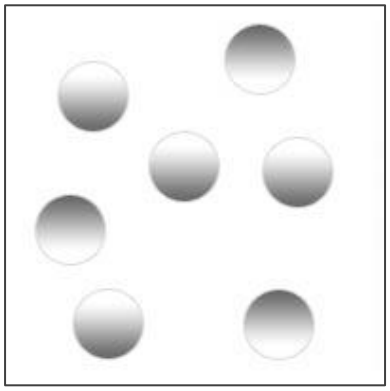


Figure 8: Protrusions & indentations (based on Ramachandran's image)

Similarly as for geometry, there are some assumptions that the brain makes about luminance. One of the best known examples is perception of protrusions and indentations (Figure 8) that depends on an orientation of the shaded disk. The reason why brain interprets disks with light top and shaded bottom as convex is built-in assumption that sun light is coming from above and convex objects are illuminated this way, whereas for concave object is illumination the other way around (Ramachandran, 2005). The size and intensity of shading then indicates radius of curvature.

When we consider illuminance of the world in general, discoveries have been made that show that human vision cells are much less sensitive to diffuse light than to spot light and that brightness is perceived in a logarithmic manner. For example, the appearance of the scene is not very affected for a human observer when sun goes behind the clouds and the amount of light reflected is reduced by a large factor (Hubel and Wiesel, p. 659). Furthermore, many experiments had been conducted concerning perception of lightness of the objects and patches with conclusions that it is relative to the lightness of the surrounding environment.

The final property of the inverse problem that needs to be considered is colour. The perception of colour tint and shade is, in common with perception of lightness, dependent on surrounding regions (Purves, 2010, p.146). By contrast, perceptual colour constancy is demonstrated via stable colour hue perception of objects under different illumination conditions. Most observers can distinguish the main colour groups (at least primary and secondary colours of the colour wheel) of the objects independent of composition of light source (Land, 1959).

When addressing perception in relation to information acquisition, the last thing that needs to be examined in processing is attention. Attention is involved in the sorting of competing visual stimuli received, as mentioned before, from lower levels in hierarchy of visual system. Because it doesn't make sense to model and render what a target audience will not notice, it is important to consider human attention character in the process. Even though attention focus is partially subjective to an individual, one possible approach to decide the question "what to prioritise" in conversion is a creation of saliency maps, which integrate different visual features that contribute to attentive selection (Niebur, 2007).

2.1.2. Projective Geometry and Image Formation

Projective geometry is often used in both computer graphics and computer vision. For information recovery from a 2D image about the layout of a 3D scene and for synthesis of 3D objects onto a 2D plane it is necessary to understand transformations defined in projective geometry and how the observer's viewpoint is formed.

There are several different geometries recognised by mathematicians with the concept of geometry defined as a space together with a group of transformations for that space (Brannan *et al*., 2012). Best known to general public is Euclidean geometry that can have a two-, three- and higher dimensional space. In Euclidean geometry points have to satisfy certain relationships – invariants, defined in terms of distances and angles, and a group of permitted transformations comprises isometric transformations (translation and rotation) that can be simply described as "a motion of a rigid object" (Hartley and Zisserman, 2004, p. 38). An algebraic approach to Euclidean geometry uses the Cartesian coordinate system to describe geometry in a numeric way, which is crucial for design of the computational methods and algorithms (Hartley and Zisserman, 2004, p. 26).

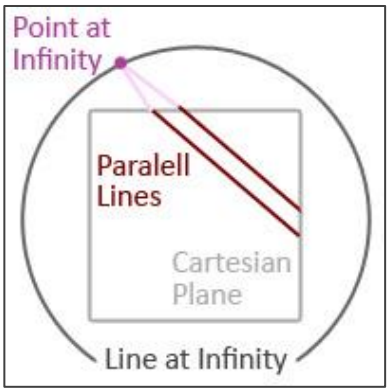


Figure 9: Projective plane (based on Wildberger's image)

Moving from Euclidean geometry towards the others, the number of valid invariants reduces and the number of allowed transformations increases. Similarity geometry doesn't preserve the lengths between object points but allows an isotropic scaling – the whole object is scaled uniformly. Affine geometry involves letting go the notion of perpendicularity, but still recognises parallelism of lines. Indifference to the angle measurements allows performing non-uniform scaling transformations. Finally, projective geometry discards parallelism by defining points at infinity (Figure 9) as meeting points of the parallel lines, which together form a line at infinity (Wildberger, 2009). All the previously mentioned transformations plus perspective projection are allowed. A downside is that few attributes from the 3D scene geometry are preserved once projected onto 2D plane (Figure 10 and 11).

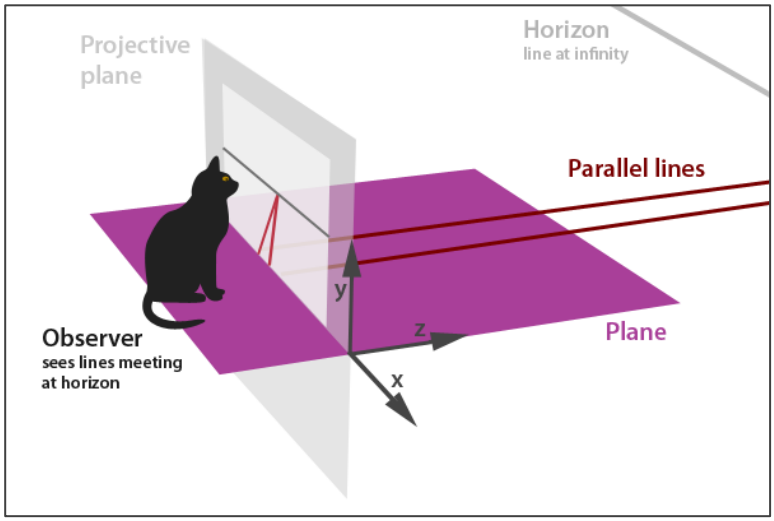


Figure 10: Seeing parallel lines meeting on projective plane

Straightness of lines is retained but lengths and ratios are different (Hartley and Zisserman, 2004). Conic sections are projected as conic sections but again it is not possible to tell whether projected object is an ellipse projected as a circle, or parabola projected as ellipse.

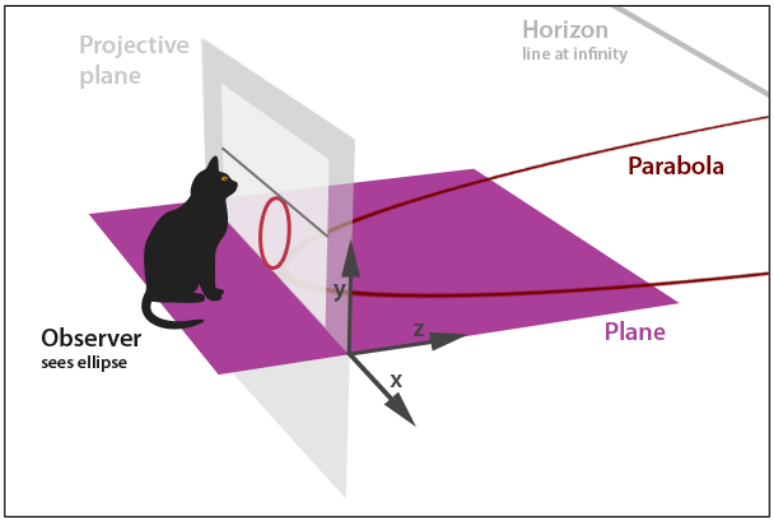


Figure 11: Seeing parabola projected as ellipse

To deal with infinity in a consistent way and to make computation of the projective transformation of one plane onto another easy, a system of homogeneous coordinates was invented to represent this transformation by a matrix. The advantage of the system is that all coordinates of points can be represented via a finite set of homogenous coordinates. It assumes a projective plane in 3D space where all lines (or planes) going through the origin of the coordinate system are intersecting the projective plane at a unique point (or line in case of planes).

In Euclidean 3D space, a position of any point A is defined by three coordinates [x, y, z], which represents how far has point moved from the origin along each axis. In projective geometry, a projective reference plane is defined as z = 1, parallel to plane z = 0, where the origin of coordinate system lies (Figure 12). With this setup, any line going through the origin and doesn't lie on z = 0 plane, is going to intersect the reference plane at a unique point. A line going through the origin and the point [x, y, z], intersects the reference plane in the point $[\frac{x}{z}, \frac{y}{z}, 1]$. This line is also going through all the points that can be obtained by scaling of these three coordinates, e.g.: [2x, 2y, 2z]. So each line going through the origin has one corresponding point on reference plane. The lines that lie on z = 0 plane give a direction for every group of parallel lines on the reference place. And since all parallel lines with the given direction are meeting at one point at infinity, each line lying on the z = 0 plane corresponds to point at infinity of the reference plane (Wildberger, 2009).

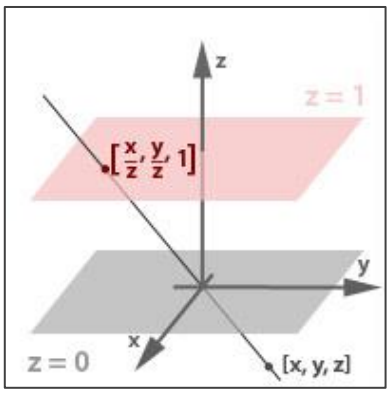


Figure 12: Homogenous coordinates

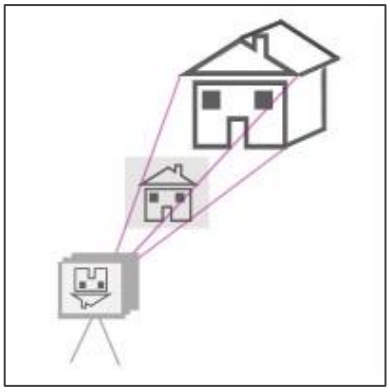


Figure 13: Image formation

Before we illustrate how the notion of projective transformation is useful in practice, we need to examine the image formation process. For the illustration (Figure 13), we use pinhole camera model, which is good approximation for an observer viewing an image. When an object is viewed by a camera, the rays of light are entering a camera via pinhole and projecting an inverted image onto imaging surface (analogically with lens and retina). With the pinhole as a centre of the projection, it doesn't make difference whether we use for calculations inverted image behind the pinhole or image with original orientation in front of the pinhole (Malik, 2012).

Objects and a camera are positioned in the world (real or virtual) and this position is defined in the world coordinates. Each object can have its own local coordinate system and camera has its own coordinate system that defines how to describe points in the world from the camera point of view (Slabaugh, 2015). Let's label a distance from the centre of projection as f. Using the camera coordinate system, we project point P [X, Y, Z] to the point p[x, y] on the image plane (Radke, 2014). Based on similar triangles rule we get equations:

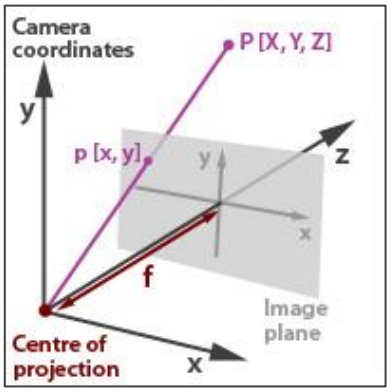


Figure 14: Point projection (based on Radke's image)

$$\frac{x}{f} = \frac{X}{Z} \qquad \frac{y}{f} = \frac{Y}{Z} \qquad => \qquad x = \frac{fX}{Z} \qquad y = \frac{fY}{Z} \tag{2.1}$$

Now we can see that if we know f and one of the parameters X, Y, Z the rest can be easily computed and position of the point in camera coordinate system determined. All these equations can be put together into camera intrinsic calibration matrix that converts points in the space to points on image plane:

$$\mathbb{C} \quad = \quad \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

As we mentioned before, every object has its local coordinate system. In order to place objects into world coordinate system, and then to camera coordinate system they need to go through series of transformations (rotation, translation). We can encapsulate that in transformation matrix:

$$\mathbb{T} \quad = \quad \begin{bmatrix} r11 & r12 & r13 & tx \\ r21 & r22 & r23 & ty \\ r31 & r32 & r33 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Now let's assume that there is a point P on a plane, which is defined in the object local coordinate system as z = 0. In that case point P has coordinates [a, b, 0, 1] (we put 1 as fourth coordinate so we can calculate with matrices). If we apply a series of transformations and projection, we should get point P proportional coordinates projected on an image plane:

$$p \sim C\ T\ P \tag{2.4}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & tx \\ r21 & r22 & r23 & ty \\ r31 & r32 & r33 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 0 \\ 1 \end{bmatrix} \tag{2.5}$$

After simplifying the equation we get:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} fr11 & fr12 & ftx \\ fr21 & fr22 & fty \\ r31 & r32 & tz \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} \tag{2.6}$$

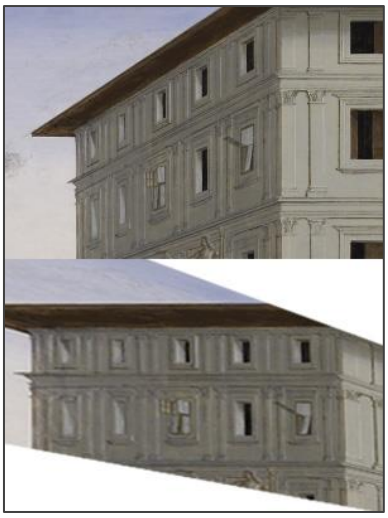


Figure 15: Removing perspective distortion

This means that for planar surfaces, 3D to 2D perspective projection reduces into 2D to 2D transformation (Collins, 2007). Being able to calculate a projective transformation of the object plane into the image plane, doing the inverse transformation returns original object plane from the section of an image plane. The inverse transformation can be computed from four correspondence points (points with known position on the image and object plane), from which no three are in one line (Hartley and Zisserman, 2004, p. 35). The practical application of the inverse transformation is to remove projective distortion from a perspective image of the plane and get building blocks for our 3D scene reconstruction. Figure 15 shows transformation of the building wall which is removing perspective distortion (notice that this particular transformation creates more distortion for the roof and other wall).

2.1.3. Perspective in Art

Understanding perspective in art is crucial for choosing an appropriate painting for the conversion. Being able to tell whether perspective projection was properly applied by the artist is necessary prerequisite for creating a scene that won't be deformed in unusual ways. Basically we can split paintings into three groups: paintings created by artists who didn't use perspective (their paintings can be interpreted as orthographic projections) or didn't

understand it properly, artists who applied a linear perspective system, which can be followed easily in their art and interpreted in certain ways, and artists who understand and use perspective in misleading ways.

Figure 16: Erroneous perspective in medieval art

The invention of linear perspective in graphic art had greatly influenced painters' approach to creation of authentic 3D spaces. Before that, medieval painters had been trying to create an illusion of 3D space via building and figure shading, changes in sizes of the objects and showing corners of the cubical objects, but on the first sight paintings seem flat and shallow and after deeper inspection spaces and geometry appear inconsistent (e.g. Figure 16). A century before the Renaissance, more precise rules (for directions of lines and planes based on eye-level and tile patterns) were invented, but they were too complicated to follow and were still producing ambiguous results under certain conditions.

Linear perspective was invented by Filippo Brunelleschi at the beginning of 15th century and it was adopted by all painters from 1500 for the next four centuries (Kemp, 1990). Linear perspective is basically a system of mathematical rules that enables anybody who follows them to produce realistic illusion of 3D space in a 2D image (Figure 17). Brunelleschi developed the system for himself in order to create proper sketches of the buildings but its application is much more general.

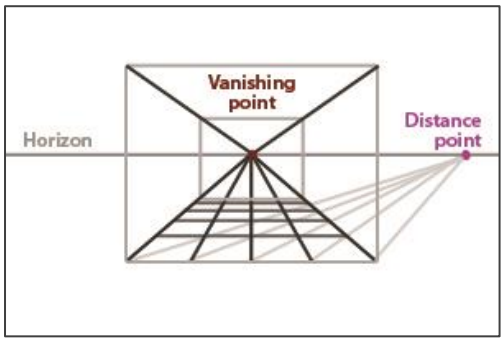


Figure 17: Linear perspective

Essentially, linear perspective system contains just few elements – a horizon line, a vanishing point, and group of orthogonals (receding parallel lines). Some of the images contain also a distance point and transversals (lines perpendicular to the orthogonals). The vanishing point lies on the horizon and all orthogonals should meet at it.

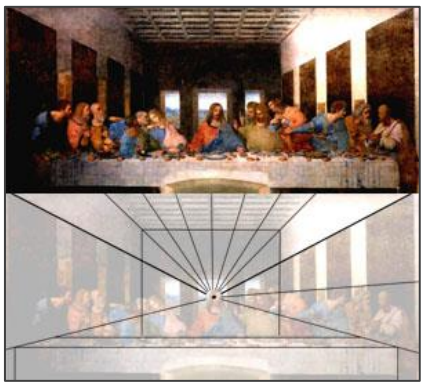


Figure 18: Finding orthogonals in a painting
(Image credits: 2Draw Wiki)

Orthogonals over the horizon incline downwards and orthogonals above the horizon do incline upwards. A distance point serves as supporting aid in order to create the illusion of proportionally decreasing regular patterns (Zucker and Harris, 2015). A similar procedure is used in paintings with two vanishing points on the horizon. For our scene reconstruction, an important thing is to find orthogonals in the painting to be able to assess level of perspective distortion and establish the spatial

relations of the objects (Figure 18).

We will not attempt to do conversion of a painting with missing, distorted or misleading perspective. Some paintings even appear logically impossible to exist in reality; however when one thinks outside of established patterns of everyday reality it is in some cases possible to find a solution that works at least from one angle (Figure 19).

Figure 19: It's all matter of perspective

Famous print of M.C. Escher was thought for long time as an impossible reality. However 3D model was created from Lego, showing one possible interpretation of the scene. (Photo credits: print - © MC Escher Official Website, photos - © Andrew Lipson)

2.1.4. Standard Computer Vision Techniques

The process of painting conversion consists of a series of tasks, some of which are well defined problems in the computer vison domain, with several available solutions. Algorithms generally work with images stored as matrices of pixels of various datatypes and colour spaces.

Noise reduction may be required prior to any other image processing. 'Salt and pepper' noise is caused by small number of pixels with very different colour intensity from the surrounding pixels. Gaussian noise is based on assumption that noise values are Gaussian-distributed. There are several procedures each with different pros and cons available to remove image noise: algorithms splitting chroma and luminance noise and treating each one separately to prevent losing image information contained in luminance, linear smoothing filters with averaging neighbour pixels values or non-linear filters replacing pixels with median value of neighbouring pixels ('Noise reduction', 2015).

Image segmentation is a process of dividing image into multiple parts that consist of related pixels. Many algorithms for image segmentation have been developed and benchmarked against human-labelled images (Szeliski, 2010, p. 269). Well-known groups include edge detection algorithms, clustering methods, level-set algorithms, histogram-based methods, region-growing methods and many others ('Image segmentation', 2015). The applicability of a particular technique depends on the image type and quality. Image matting – a technique of removing foreground objects from background with separation of blended edges (Szeliski, 2010, p. 105) would for example only be relevant if we do a very precise modelling of 3D object from the large source.

Feature detection is typically used for image matching to find correspondence points or for performing object instance and category recognition but in our case it is needed for marking points of interest for a human spectator, that need to be focused on when performing conversion. Taking into account the presence of a human spectator, the solution can be a combination of standard feature detection procedures and creation of a salient features map that that considers properties of human vision (Itti and Koch, 2000).

Other techniques of computer vision that are potentially useful for our conversion include in-painting (to reconstruct missing and damaged parts of the image), image resampling (to enlarge small objects), and image enhancement (to remove shades from textures, or change texture lightness).

2.2. Modelling and Synthesizing

OpenGL is a hardware-independent application programming interface consisting of hundreds[2] of distinct commands that can be used to define the objects and operations needed to produce interactive 3D applications (Shreiner *et al.*, 2013). OpenGL 4.x together with OpenGL Shading Language (GLSL) and related libraries (to take care of windows or to import mesh models) provide a wide range of functionalities useful for painting scene synthesis.

OpenGL is an interface to graphics hardware, which is programmable through shader programs (written in GLSL). The created software runs in a client/server mode, where the OpenGL application, acting as a client, loads data (meshes, textures, shaders) from the persistent storage and then transfers it to the GPU, which acts as the server. Shader programs are then executed on the GPU. There are several types of shader programs: vertex shader, fragment shader, geometry shader, and two types of tessellation shader that process data at different levels of granularity (Slabaugh, 2015).

A typical graphics rendering pipeline (Figure 20) starts with processing of raw vertices and primitives via vertex shader and then transformed vertices go through a primitive assembly stage in order to organise them into their associated geometric primitives. After that geometric primitives are sent to the rasteriser for fragment generation and subsequently fragments are processed by the fragment shader, where they are assigned their final colour. At the end, all fragments are tested and only visible fragments are put into final image that is shown on screen. Tessellation and geometry shaders may optionally be used between the vertex shader and primitive assembly stages (Shreiner *et al.*, 2013).

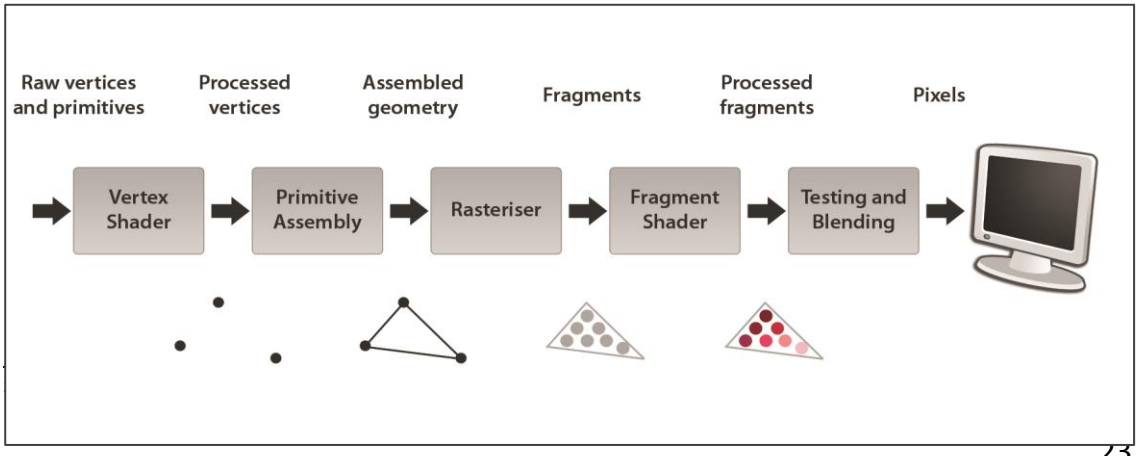


Figure 20: Graphic rendering pipeline

2.2.1. Camera

To simulate real world projection and viewing, a synthetic camera model was created for OpenGL that uses transformations between different coordinate systems to render an image. These transformations are implemented as matrices (illustrated in Section 2.1.2.) and OpenGL has defined functions for each one of them. As in the real world, it is possible to move the camera based on user input or create a predefined camera track.

The camera has a position (in world coordinates) and its viewing direction is defined by the point to which it is oriented (also in world coordinates) and orthogonal 'up' vector (information needed for rendered image where is up). Besides perspective projection, OpenGL also offers the possibility of an orthographic projection. For the perspective projection (see Figure 21), it is necessary to define vertical viewing angle (fov), the aspect of the viewing window (w/h), distance of a near clipping plane from centre of projection (camera position), and distance of a far clipping plane. This creates the view frustum in which all objects rendered into the final image lie (Slabaugh, 2015).

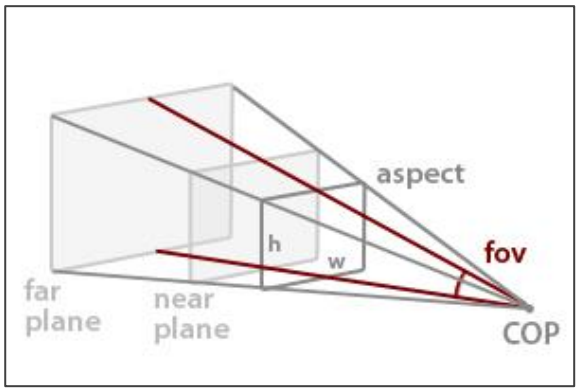


Figure 21: Perspective projection

2.2.2. 3D Mesh Models

OpenGL and related libraries support loading of 3D mesh objects, which can be placed in virtual environment with position specified in world coordinates. Despite the possibility of creating textured 3D objects straight via OpenGL API, it would be too difficult for most of the objects, and so mesh objects are created in specialised software tools like Blender.

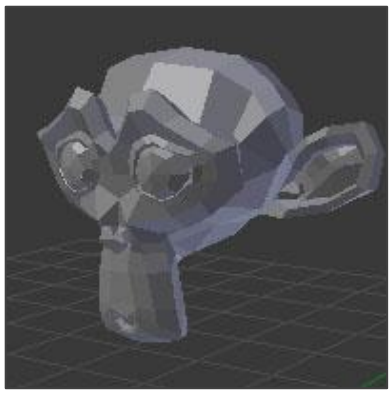


Figure 22: Example of 3D mesh object

A 3D mesh object (Figure 22) is collection of vertices, edges and faces that form together a polyhedral body. Faces are most often quadrilaterals but sometimes also triangles. Colours, textures and materials are assigned to different parts of the mesh during the modelling process and loaded when the final object is rendered in OpenGL environment.

Meshes can be of different quality; generally they are categorised as low-poly and high-poly meshes. Low-poly meshes count polygons in thousands, high-poly meshes count in hundreds of thousands and are often made by applying surface subdivision on simpler models to obtain a smooth appearance. For real time rendering, low-poly meshes are used to keep acceptable performance. But it is possible to improve the appearance of a low-poly mesh via applying textures made for the same mesh with more surface subdivision.

2.2.3. Skybox

Creating a skybox (Figure 23) for a virtual environment or a computer game is common way of modelling the background environment. The basic version of a skybox is just mapping six square textures on to a large box surrounding the camera. A texture for the six images needs to be properly designed to avoid any seams. When the camera translates, the skybox moves with the camera. This gives an impression that skybox is very far away and therefore not moving in the manner of close by objects (Slabaugh, 2015).

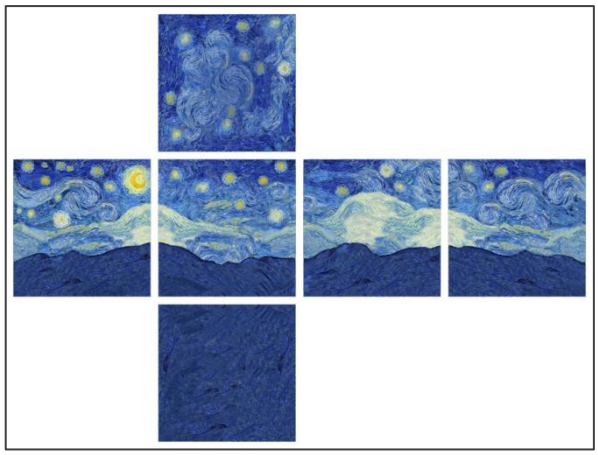


Figure 23: Unfolded skybox texture

A skybox can be also programmed as a cube map. Visually it looks exactly same as when it is done by simple six separate textures mapping, however the texture from cube map is accessible for further interactions with objects, such as being reflected from shiny surfaces (Wolff, 2011, p. 123).

2.2.4. Terrain

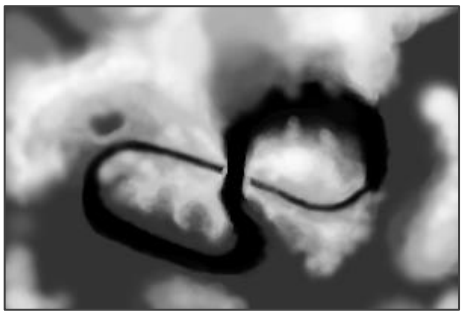


Figure 24: Heightmap

Terrain or parts of terrain in virtual environments are frequently designed as 3D mesh models; however OpenGL provides generous options for other types of modelling. The simplest solution how to model terrain is creation of a flat plane with one or more seamless textures. Textures are mapped to plane coordinates in fragment shader, where texture blending is set up as well (Wolff, 2011, p. 113).

More advanced alternative to terrain modelling is the use of heightmaps (Figure 24). A heightmap is a matrix of unsigned char variables (which are different shades of grey in a greyscale image) that define height values for the terrain. When applied, it transforms a flat plane into natural hills. (Polack and LaMothe, 2003, p. 16). When applied in combination with multi-texturing (Figure 25) the result is a very organic looking terrain.

Figure 25: Terrain with multiple textures

2.2.5. Lighting

The lighting model in OpenGL tries to efficiently mimic real world light interactions. It does not do this in a completely accurate way due to the complexity of real light interactions and multiple light reflections. However, it provides several different light modes, which when added together give quite realistic results.

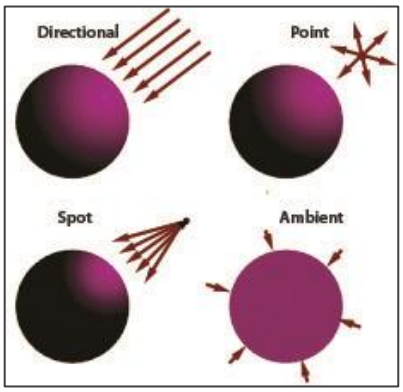


Figure 26: Different lighting modes (based on Slabaugh's image)

A directional light source is global and it is considered to be infinitely far away. Light rays are parallel when they hit the object. A point light source is emitting light in all directions from single point and it is strongest next to the source and getting weaker with increasing distance from the source. A spot light is similar to a point light but it emits light in one direction in the shape of cone. Ambient light is uniform low level lighting that should account for all light reflections from various surfaces (Slabaugh, 2015).

There are three types of reflectance based on different object surfaces: ambient, diffuse and specular. A reflectance model is a mathematical description of how incoming light is reflected from a surface. With ambient reflection, a resulting colour depends only on light's ambient component and surface ambient material. Therefore an object is uniformly illuminated across the surface. For diffuse reflectance, a brightness changes with angle between surface normal and direction of the light source. However, it appears equally bright from all viewing angles. Specular reflection produces highlights, which can be observed on shiny surfaces. Besides the angle between light source and surface normal, it depends also on the viewer position (Slabaugh, 2015).

A reflectance model can be calculated in the vertex or fragment shader. For many scenes, there are significantly fewer vertices than fragments, so per-vertex lighting calculation is faster. However, since light is only interpolated along primitives (edges), the resulting shading lacks smoothness.

2.2.6. Special Effects

OpenGL offers a wide range of special effects that can be applied to a rendered scene. Even though OpenGL implements a local lighting model, there are several ways to model shadows. For planar shadows, occluder primitives are projected from a light source onto a receiver plane, with shadow mapping method, shadows are treated as a problem of visibility from the light source. A mirror-like reflection of parts of the scene can be achieved via multi-pass rendering, when scene and reflection are rendered separately and blended at the end. Different versions and degrees of blur can be programmed in the fragment shader by averaging neighbouring values of pixels. A fog can be created by blending colour (or texture) with original colour of rendered scene as a function of distance from the camera (Slabaugh, 2015).

2.3. Existing Research and Tools

2.3.1. 2D to 3D Conversion and Mesh Creation

One of the crucial problems of converting a 2D image to a 3D scene is to determine appropriate positions of the individual elements in a scene when adding depth. Make-3D is a tool that resulted from the research of depth reconstruction from a single still image (Saxena *et al.*, 2007) followed by 3D scene structure estimation (Saxena *et al.*, 2008). The focus of the research was on unstructured environments (no special assumptions are made about the structure of the scene). The method of conversion consisted of image segmentation into small, approximately planar surfaces (similar to graphics representation of objects in OpenGL) and determination of 3D positions and orientations using a machine learning algorithm. Thereafter, a 3D mesh model of the scene could be reconstructed. Based on the videos of 'fly-arounds' results of created 3D models, the tool has been tested in this project but we didn't use its outputs at the end.

Most of the research concerning 2D to 3D transformation has a narrower scope and is focused on specific techniques of conversion. A novel approach was introduced with the alignment of 2D depictions, including paintings of architectural motifs (e.g. city square with monument), with the 3D model of the site (Aubry *et al.*, 2014). It solves the problem of painting alignment, with many elements in the scene and created in variations of artistic styles, via small sets of visually discriminative elements. The disadvantage of this method is the need for a priori 3D models of the buildings from the painting, so it would be useful for our research if our painting contained only known places for which 3D models of the landmarks already exist. When having more than one image source, effective methods have been developed for architectural scene modelling (Debevec *et al.*, 1996).

Using much simpler existing 3D models, some research has been done in creating object models from photographs by aligning and joining visible parts of the object from in an image with similar 3D models to recreate the non-visible parts (Kholgade *et al.*, 2014). Possible mismatches between the image and the model are corrected using pixel information from the image while keeping approximate object symmetries and also considering illuminations. The practicality of using this approach in our research depends on the availability of relevant 3D models.

In general, techniques for mesh creation vary depending on the complexity of surfaces, textures and object realism. Promising research has been done in many areas concerning object 3D modelling, including an automatic gradient mesh generation for whole images (Lai *et al*, 2009), significantly speeding the process and removing the need for manual interaction. Simple 3D shapes can be extracted from a single image source by 3-Sweep tool, with human user segmenting complex objects into their components and positioning them in space. After that algorithm reshapes the component to fit the image of the object and satisfy constraints given by user defined geometry (Chen *et al.*, 2013).

In parallel, advanced tools for 3D modelling have been developed. Gimpel3D is both versatile and free and it enables user to reconstruct perspective spaces and do image projections onto 3D models (Gimpel3D, 2015). Disadvantages of this tool are obscure formats of the final models and outdated installation that may not work with newer operation

systems. The most popular tool among 3D artists is Blender that has ongoing community support and it has been used in this project. More about Blender features can be found in next chapter.

2.3.2. Painting Conversion

In their publication Multiple View Geometry in computer vision, Hartley and Zisserman shows creation of 3D model of the room created from a single painting (Hartley and Zisserman, 2004, p. 10). They reveal that several assumptions had been made when reconstructing the scene and their process builds on knowledge of linear perspective in painting.

Zisserman had also cooperated with art historian M. Kemp and researcher A. Criminisi on conversion of Masaccio's Trinity painting into a 3D model (Criminisi *et al.*, 2005). Besides that, they present in their paper techniques for an assessment of perspective accuracy and geometry consistency.

The artist M. Cauley converted Van Gogh's Night Café painting into a 3D room, focusing more on aesthetics of the painting, than precise mathematical reconstruction (BBC News, 2015).

2.3.3. Image Reconstruction

Since the source of information about a 3D scene is a single image, by isolation and extraction of the foreground objects into layers, missing parts of the scene in the background will become visible once objects taken away. Therefore it is necessary to produce replacements for missing parts of various items in the image. For this purpose, many algorithms to find nearest neighbour matches of missing parts have been developed, some of them obtaining efficiency through random sampling to find good matches and their subsequent propagation (Barnes et al., 2009). In the area of recreation of larger images from smaller samples, successful research has been conducted for more than 10 years (Kwatra *et al.*, 2003).

2.4. Development and Production

2.4.1. Software Development

The Rational Unified Process (RUP) is an iterative-incremental software development framework that is adaptable to the needs of organisations and projects. RUP defines four project life-cycle phases: Inception, Elaboration, Construction and Transition. Each phase can be split into more iteration and has one or more defining milestones (Shuja and Krebs, 2008).

RUP methodology also includes disciplines, which are categories for activities based on major areas of concern. There are six core disciplines that are directly linked to software engineering activities and three supporting disciplines that are concerned with management and structure of the project (Shuja and Krebs, 2008).

Core RUP disciplines:

- Business Modelling
- Requirements
- Analysis and Design
- Implementation
- Test
- Deployment

Supporting RUP disciplines:

- Configuration and Change Management
- Project Management
- Environment

RUP also involves detailed definition of the roles and artefacts but considering the scope of this project, only several key artefacts will be used in the process.

Rapid application development (RAD) is an approach that puts less emphasis on planning tasks and pays more attention to development. It emphasises the necessity of adjusting requirements in the process and sometimes it uses prototypes in place of design specifications. RAD is well suited for developing software that is driven by user interface requirements (‘Rapid application development’, 2015).

2.4.2. Production Pipeline

There are several different production pipelines established by game developers and artists in order to produce artwork or animated application. The production pipelines were developed because of the scope of work and number of people involved in animation production but it is also a useful framework for the single artist/developer (Pellacini, 2009).

Many variations of what the production pipeline should encompass have been described. One of the examples suggests (Boudon, 2013):

- Pre-production
- Modelling
- Painting and texturing
- Rigging
- Animation
- Dynamics
- Lighting
- Rendering

For the project, animation and dynamics stages won't be part of the production pipeline.

2.4.3. Web Design and Development

Hundreds of books have been written about web design and development from all possible angles and to keep track of all trends is hardly possible even for full time web developers.

From the technical side of the topic, a shift towards HTML5 standard occurred in last couple of years, which together with CSS3 provides much wider options of semantics and styling to web developers. HTML5 and CSS3 code can replace some JavaScript code and provide the same functionality. However JavaScript with its libraries still play a major role in modern website code. PHP retains its presence in templating and web frameworks.

From the user perspective side, usability testing is inseparable from development of any medium size website. Krug summarised important principles of web usability in his book Don't Make Me Think, which is an understandable guide for any beginning developer (Krug, 2006).

3. Methods

This chapter focuses on tools and techniques used in the course of this research project. First, it presents the development environment, software and relevant libraries used in the project. Then it provides a methods overview, followed by their description in the individual project stages. The main stages of the project were planned according to the RUP phase sequence for application development, but other tasks of project (web development and artistic production pipeline) are accommodated by this structure as well.

3.1. Software Tools and Programs

3.1.1. OpenGL

OpenGL 4.0 API with its libraries for Windows together with C++ was chosen for application development. The application was developed on the Windows platform, using the Microsoft Visual Studio 2015 IDE. Shader programs that are used in conjunction with the OpenGL to process a colour of the image were written in OpenGL Shading Language. The application was built on top of the OpenGL template for INM376 Computer Graphics module.

OpenGL was chosen for the following reasons:

- OpenGL is a platform independent API.
- Implementations of OpenGL can be easily made cross-platform.
- OpenGL is an open standard, which simplifies portability and lowers costs for the developer.
- OpenGL has bindings for a variety of programming languages, including C++.
- Complexity is not overwhelming, and is consistent with the complexity of the project.
- Modern OpenGL has low driver overhead.
- Detailed documentation and many tutorials are available.
- OpenGL is extensible, so new hardware features can be made available quickly.

3.1.2. Matlab

Matlab 2015a has advanced image processing and computer vision toolkits and therefore is ideal for some tasks in the proposed production pipeline. Matlab is a high-level technical computing language that makes it easy to test different algorithms and optimise their parameters. With Matlab it is possible to create many outputs quickly and then assess their quality accordingly. Much academic research code (Make3D, Saliency maps) is runnable only via Matlab. The downside of Matlab is the lower efficiency of the generated algorithms and for the long term use only paid license is available.

3.1.3. OpenCV

For more complex image processing algorithms, the OpenCV library was preferred over Matlab. OpenCV is an open source computer vision library written in optimised C/C++ that provides interfaces to C, C++, Java and Python languages and supports Windows, Linux, Mac OS, iOS and Android. For this project OpenCV was used with the C++ language on the Linux platform.

Modern OpenCV has several relevant properties for our project:

- It is much closer to machine language than Matlab, so runtime processing is more efficient.
- The C++ interface enables object oriented programming and solves the problem of memory management that was an issue of earlier OpenCV version based on C.
- It has a larger collection of image processing functions than Matlab.
- It works with various image formats and different colour channels.
- Step by step tutorials and a large community that participate on support forum makes it easy to get started programming.

3.1.4. Blender

Blender 2.75 is a professional open-source 3D graphics tool. Blender enables the creation and export of models into several different formats that can be imported by OpenGL to a virtual environment and rendered with the rest of the scene. Blender was chosen because it contains all important modelling features and unlike Maya or 3ds Max is free of charge.

Blender has following features pertinent to our 3D modelling:

- Tools such as subdivide, extrude, loop cut and inset that enable rapid creation of new geometry.
- Different modes of unwrapping models for texture mapping.
- Various types of texture creation including cloning and baking.
- Inserting an image as a plane option that prevents difficulties with texture mapping.
- Several modifiers that speed up modelling, such as array modifier for creating and arranging repeating objects, bevel modifier for cutting off edges, mirror modifier for modelling symmetrical objects, subdivision surface modifier to smooth object surface and many more.
- It contains many templates of basic geometric shapes, but also complicated geometry such as terrain or trees.
- Sculpting, rigging and skinning tools are available to create non-geometric objects.

3.1.5. Adobe Photoshop

Adobe Photoshop CS3 is professional image processing software that has many useful tools, such as selection tools, clone stamp tool, liquify tool and multiple layers that can be manipulated, merged and blended. Adobe Photoshop was chosen solely because of the present installation at the beginning of project and familiarity with its interface. From the tools point of view, Gimp is equally good alternative for free.

3.1.6. Web Tools

A combination of several web technologies and tools was used for development of the Conversion User Guide. For an interactive part of the website, the WordPress content management system was chosen for its simplicity and template versatility. The rest of the website was created in HTML5 and CSS3, with PHP code for page templates and JavaScript code for a scrolling menu, menu changes as part of responsive design and a carousel of images.

3.2. Methods Overview

3.2.1. Application Development

Based on the RUP methodology (Shuja and Krebs, 2008) but with respect to the nature of this project (single person, non-commercial), the following activities were conducted in the course of the project:

- Business modelling
- Requirements capture
- Analysis and design
- Implementation
- Testing
- Deployment

3.2.2. Production Pipeline

The production pipeline shows the process of creation of individual objects (skybox, terrain, mesh models) from painting. The production pipeline activities in this project are slightly different from the typical tasks of a visual artist's workflow, given by the scientific nature of the project and the fact that all objects are integrated in an OpenGL application where lighting and special effects are globally applied. In this project, the production pipeline includes:

- Image processing
- Image analysis

- Camera calibration and initial layout
- Painting and texturing
- Modelling and rigging

3.2.3. User Guide and Web Development

The Conversion User Guide as part of a website was developed in a RAD mode ('Rapid application development', 2015). The main tasks were the following:

- Requirements capture
- Information architecture outline
- Wireframe design
- Functional prototyping
- Graphics Design and Style Creation
- Template creation
- Functionality programming
- Content supply
- Promotion

3.3. Stage 1 – Inception

The inception stage of the project took place prior to the project proposal approval. During this stage a business case for the project was established with its aim, objectives, products and beneficiaries. The current status of the research in the selected areas was assessed and a research question was formed based on that knowledge. Initial estimations of required technology for development were made and the project plan together with an Elaboration stage plan was outlined. The project proposal also indicated what main project products and sub products would be and several key requirements. This stage finished with the project proposal submission.

3.4. Stage 2 – Elaboration

The elaboration stage started at the beginning of May with a milestone at the end of June. Prior to starting any work, a detailed analysis of requirements was needed done in order to understand the properties of the project deliverables. An initial blueprint was recorded in the first version of the Project Product Description (Managing Successful Project with PRINCE2™, 2009). The Project Product Description (Appendix B) provides a breakdown of the project products (Section 1.3.) into finer sub products and contains a description of each product, its form, requirements that the product must meet (quality criteria), technology and

development skills needed for its production and its dependency on other products. Subsequently, Analysis and Design were performed for the application and the production pipeline work was forked, with image processing and analysis tasks finished in this stage. For the Conversion User Guide and website, information architecture was specified and wireframe design produced.

3.4.1. Application Requirements Capture

The requirements discipline according to RUP "consists of activities that ensure effective requirements engineering and management" (Shuja and Krebs, 2008, p. 93). At the end of the project, it must be possible to verify whether outcomes fulfil the requirements defined at the beginning of the project. Because of the character of the application and its very straightforward user interaction of "changing the speed of the tour and setting different camera views" (Section 1.3.), the main emphasis of capturing requirements was on analysing the problem and defining the system (Shuja and Krebs, 2008, p. 95).

As mentioned in Chapter 2, the problem to be solved spans both computer vision and computer graphics. Although at completion of the Computer Graphics module an understanding of tools and techniques in computer graphics was sufficient for the project, understanding of problems and solutions in computer vision was very limited. Therefore, several online courses on computer vision and image processing have been taken, including Computer *Vision* (Malik, 2012), *Computer Vision for Visual Effects* (Radke, 2014), *Intro to Digital Image Processing* (Radke, 2015) and books examined, including *Computer Vision: Algorithms and Applications* (Szeliski, 2010), and *Multiple View Geometry in Computer Vision* (Hartley and Zisserman, 2004). To build up knowledge about the modelling component of computer graphics, a series of modelling courses were taken at CG Cookie learning portal (CG Cookie, 2015) that also helped understanding production workflow.

After finishing the first round of the knowledge gathering process, the first version of the Project Product Description (Appendix B, CGA identifier) was created by breaking down the application product into sub-products. Even though it was possible to specify several requirements for individual sub-products and a basic sequence of activities for the production pipeline (Section 3.2.2), the whole object production process involved significant uncertainty. This was partially caused by the nature of the problem of computer vision that needed to be solved (with no "correct" solution) and partially by lacking human input even though a human audience judge the result application at the end.

For this application, analysis of stakeholder needs didn't apply, because it wasn't an understanding of needs that would determine the success of the application. Instead, problems in the human vision and perception domain were judged more important. Thus, a thorough examination of these domains was conducted in order to extract information that, when applied to the production process, could reduce uncertainties of conversion of a 2D image into a plausible 3D scene. After this iteration, the Project Product Description was refined accordingly.

The last piece of expertise that had to be taken into account in our problem analysis was the nature of painting itself. Unlike photorealistic images of the real world, on which computer vision theory is focused, paintings can sometimes contain an image of an impossible 3D reality. A theory behind how painters created the space in their depictions has brought more useful information in the scene conversion process. Besides that, modelling difficulties connected with different styles and objects was considered when criteria for selection of the suitable painting were formed.

With basic requirements for the application, its components, and intermediate products specified in the Project Product Description, analysis and design of the system could proceed. However, in the course of analysis and design activity, several new requirements were captured and existing ones were refined.

3.4.2. Analysis and Design

The main purpose of an Analysis and Design discipline is to establish "a feasible vision for the system and to assess appropriate techniques for building the solution" (Shuja and Krebs, 2008, p. 111). An initial architecture of the application was based on the OpenGL template for the INM376 Computer Graphics module with main application class, setup classes for timer and windowing system, application classes for camera, skybox, planar terrain and mesh import, and support classes for shader programs, textures, matrices and vertex buffer object (Appendix C). Two shader programs have been used to do shading, with the lighting model applicable in the both vertex and fragment shader. Since the core version of modern OpenGL with the programmable pipeline has been used, rendering happens in a retained mode when passed vertices are stored in VRAM on the GPU in a container called a vertex buffer object (VBO) before they are rendered. Prior to rendering to a window, the window must be configured via pixel formats according to rendering requirements. (Slabaugh, 2015).

The Application class contains of three main methods, where the whole functionality is defined: *Initialise(), Render()* and *Update(). Initialise()* method runs once at the beginning, Update() and Render() run repeatedly in a loop. In the *Initialise()* method all application objects are created, the application window size is set up and used for projection matrix initiation, and shader programs are loaded. In *Render()* method, a matrix stack[3] is defined and assigned values (identity matrix when defined and then to camera view matrix is stored on top), textures are sent to samplers and the lighting is set up. Then, the individual objects are positioned in the world coordinates (applying transformations such as translation, rotation and scaling) and in the vertex shader they are transformed via the camera view matrix and projection matrix. Vertex normals are transformed as well and the resulting colour is computed for each vertex. In the fragment shader, the colour is interpolated across primitives and assigned to each fragment. The *Update()* method runs repeatedly with the *Render()* method and the camera is view updated to reflect its motion.

[3] A matrix stack is a simple way to efficiently apply composed transformations on rendered objects. It allows saving current matrix, performing some transformations and then returning to the saved matrix (Slabaugh, 2015).

This initial setup would be sufficient for integrating individual components during the development. For the final application a class for virtual tour track must to be added. Other application classes might be needed, such as for a heightmap terrain (Section 2.2.4) and cube map (Section 2.2.3), based on the type of painting to be converted. If multi-pass rendering[4] is required for various effects, the *Render()* method would require a modification and class for an additional buffer (stencil or framebuffer) to be included. The shader programs would need to reflect all these changes and additional pair of shaders might be incorporated.

Apart from the structural modification of the application, what needs to be addressed in Analysis and Design are application performance attributes that are going to be monitored and optimised in the development process. For real time rendered applications, a key performance indicator to track is frame rate. Frame rate is a number of images that are rendered per second (indicates show quickly are unique consecutive images produced). While a top boundary for frame rate can be limited by synchronising frame rate with screen refresh rate, if the frame rate is too low, the illusion of motion in our virtual tour would be impaired and user impression disturbed. A reasonable frame rate target is between 30 to 60 frames per second (Slabaugh, 2015).

There are several strategies to speed up an application. From hardware perspective, built-in shader program functions should be used, unnecessary shader programs should be removed and lighting operations should be focused in vertex shader (for applications containing simple meshes with low polygon count).

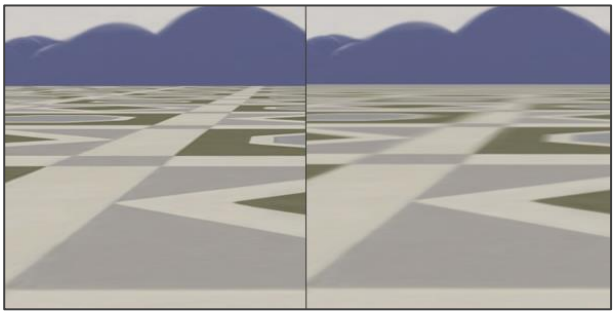


Figure 27: Effect of anisotropic filtering (on and off)

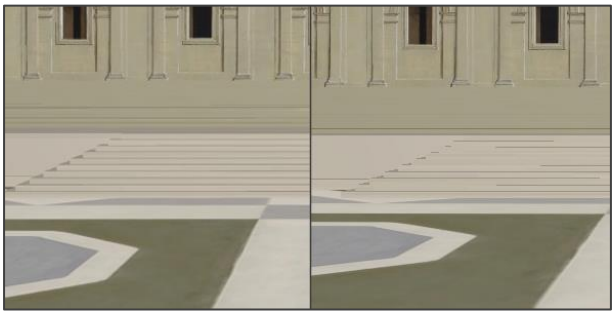


Figure 28: Effect of anti-aliasing (on and off)

From the software point of view, it is important to keep number of passes in rendering to minimum, enabling back-face culling[5], turning off anisotropic filtering[6] (Figure 27), turning off anti-aliasing[7] (Figure 28), and keep the number of polygons as low as possible. The number of polygons, sometimes called a polygon budget, will be directive for

[4] Multi-pass rendering is used, when scene or part of the scene needs to be rendered more than once (usually with some changed parameters) and then combined together in one image.

[5] Back-face culling is a process of determining whether object's polygons are front facing (with anti-clockwise polygon points order) or back facing (clockwise order), and rendering only front facing polygons.

[6] Anisotropic filtering is technique to improve texture quality when the texture is not viewed orthogonally (it prevents texture blur in distance).

[7] Anti-aliasing refers to number of techniques that prevent coarse edges and missing details of objects and textures, which are caused by inability to capture small details due to insufficient resolution

the several stages of our production pipeline and we need to make a reasonable estimation of it for our production machine.

To do polygon budget estimation, we set up variations of rendering context and environment and tested it with different polygon loads (Results in chapter 4). In the first round of testing assessed the impact of each feature on the overall performance. Based on the results we decided to turn on multi-sample anti-aliasing and set it to value four (which generates 4 fragments with slightly different sub-pixel locations and averages the result) and we have allowed anisotropic filtering for the planar terrain. Then we compared different polygon load for a single- and two-pass rendering and for two versions of lighting calculations, in vertex shader and in fragment shader. Our target rate was around 70 frames per second (we wanted to keep a reserve for additional burden, since our setting was simplified in comparison with real application).

From the results, the lighting was decided to be calculated in the fragment shader and we have set upper limit of 250,000 polygons in case of two-pass rendering and 500,000 in case of single pass rendering. Against these numbers, all models created in production pipeline need to be assessed. Another parameter to keep track of while creating models is total size of textures used in the application. With graphics card capable to store around 1GB of textures, it is necessary to optimise their use.

Now, with the production pipeline defined, a workflow for application creation was at the end of Analysis and Design split into two branches. One branch covers design and implementation of the application in OpenGL and the second branch covers the production pipeline of the individual components created and modelled from painting that will be assembled together in the application.

3.4.3. Image Processing

Our production pipeline starts with processing of the painting image source. There are many painting images available in the public domain and high resolution scans are available to download. Adobe Photoshop enables the editing of even very large sources (up to 200MB) and therefore it is always best to work with the images in their original size. The first step for some painting sources is to remove damaged edges, where the brush strokes end and canvas is visible. That can be done by simple clipping in Photoshop. After clipping the edges, optional task of noise filtering follows. Unless noise consists of small points of high contrast against the rest of their neighbourhood (so called salt and pepper noise), it is not recommended to do much filtering at this stage of the production, otherwise the texture quality intended by artist might be degraded.

To remove salt and pepper noise, the Matlab Image Processing Toolbox provides the function of median filtering that replaces each pixel value from original image *A* with the median value of its *m x n* neighbourhood (Matlab, 2015):

```
B = medfilt2(A, [m,n])
```
(3.1)

Figure 29: Box Segmentation

After these initial operations we would keep the resulting image as a reference and for the further work we would use its copies. Next, the image segmentation needs to be done to separate all objects into their respective layers. This can be done in two different ways.

Simple box segmentation (Figure 29) can be done for the objects that will be modelled according to image and their texture will not be orthographically projected onto them. Clean object segmentation (Figure 30) that removes all pixels that don't belong to object is necessary when we want to preserve as much of the object texture as possible to use it for orthogonal projection onto object 3D model.

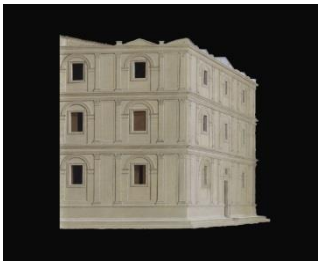


Figure 30: Clean Segmentation

There are several different methods for doing segmentation. One of them is to use the Matlab active contour method to segment grayscale image *A* into a binary image *bw* of foreground (object) and background (rest of the image), with the *mask* specifying the initial state of an active contour and *n* maximum number of iterations over which curve is evolving (Matlab, 2015):

```
bw = activecontour(A,mask,n)
```
(3.2)

A different method of image segmentation is colour thresholding, when only regions of image with certain colour threshold are cumulatively selected and the final region is separated from the background.

A special kind of segmentation must be done for the image region that becomes part of the skybox. If terrain is flat or it is a sea surface that is touching a horizon, the cutting line is given by the horizon. To find an ideal line at which to cut off the skybox region for non-trivial terrain, monocular distance cues provides useful hints and aerial perspective in particular (Section 2.1.1). We do separate blurry and blue coloured parts of distant terrain and objects to be a part of skybox texture.

The last step in the image processing activity is missing segment reconstruction, caused by foreground objects occluding the objects behind them. As a guide for the missing segments reconstruction we imitate via inpainting the perceptual filling-in of the brain (Section 2.1.1), which has been described for various combinations of geometry and textures (Sacks, 2010) (Ramachandran, 2005) in combination with the regular geometry completion and symmetry.

3.4.4. Image Analysis

The purpose of an image analysis in this project is to extract information from the image that can be potentially useful in modelling the scene. The first important feature to distil is colour. To get an image palette, we designed an algorithm that reduces the image colour space and creates a palette with selected number of most frequent colours sorted by their proportional representation (Appendix D).

The next characteristic that is useful to extract is texture. Texture quality assessment might be part of texturing activity when new texture images are created from old ones to see whether they share common characteristics (e.g. similar values of pixel contrast indicate that we didn't lose sharpness of blurriness in texturing process). The Matlab Image Processing Toolbox includes several texture analysis functions that provide information about local variability of the pixels intensity values. For example, the function for range filtering returns an array of pixels where each pixel contains the range value of the specified neighbourhood pixels. For smooth textures, the range of the values will be a small number, for rough textures the number will be larger (Matlab, 2015):

```
J = rangefilt(I, NHOOD)
```
(3.3)

Another set of information can be extracted via linear perspective analysis (Section 2.1.3) of the image. Superimposing a new layer over the image with highlighted extended orthogonals meeting in the vanishing point provides us with a framework for object positioning. With the orthogonals established we can add horizontal and vertical traversals, which divide the space into rectangular cuboids. If we can find four points that form a square on the ground plane, we can also find the distance point on the horizon.

Having the distance point enables us to partition the ground plane to equally wide stripes parallel with horizon line that give an impression of thinning with growing distance from the observer. In combination with the monocular relative height distance cue (Section 2.1.1) we can then approximate objects' layout on the ground plane. With the objects' layout set and reasonable assumptions about objects orientations, we can also calculate relative sizes (Section 2.1.2).

The last fact that we would want to know about the image is where the attention of a person observing the image will be focused. That might be useful for giving us a guideline on what to concentrate when completing individual object models, how to balance the polygon budget for individual models and at the end for a path planning. For that purpose, we used the Saliency Toolbox that computes the saliency map (Section 2.1.1) of an image (Saliency Toolbox, 2015).

With all the information extracted we can also combine different elements (colours, textures, objects) into a moodboard to better understand painting's aesthetics.

3.4.5. Capturing Requirements for User Guide

Unlike the application, which was quite precisely specified in the project proposal, the Conversion User Guide specification was left open in order to spend more time researching requirements. The first major decision relating to the guide was what would be its form. Here, the stakeholder needs come first. After conducting a search on current learning trends, materials and environments (BetterExplained, 2015) (LearnCpp.com, 2015) and conducting a brainstorming session, the following initial requirements for the Conversion User Guide were formulated, namely it should

- Be easy to find

- Contain short separated learning blocks
- Contain examples and illustrations
- Contain useful links throughout the content
- Be extensible

After considering these requirements, a decision was made that the Conversion User Guide should be in the form of an online blog rather than a more traditional published PDF document. Placing the Conversion User Guide online, it would be useful to publish scene elements produced in this project (skybox, textures, heightmaps, and models) to be reused on the same website. Likewise, with a user evaluation to be done as a part of the project, a website can embed evaluation questionnaires as well. Taking into account all above mentioned functionality, a simple use case model (Appendix E) was developed to model all the important aspects of system behaviour.

Based on the use case model, a requirement for a simple publishing system for learning articles with discussion feature became clear. On the other hand publishing different scene elements (skybox, textures, or models) would be better served with different layouts of pages. Creating a questionnaire and collecting answers would also be rather problematic with standard content management systems. Therefore it was decided that the Conversion User Guide part would be better built on the top of the WordPress content management system in form of a blog and the rest of the website would be custom-made in a combination of HTML, CSS, PHP and JavaScript code.

Having decided on the separation of underlying technology, but with a common design, the Conversion User Guide and the rest of the website were further broken down into sub products and recorded with respective requirements in the Project Product Description (Appendix B, CUG identifier). Based on the RAD approach, no further analysis is necessary and after outlining information architecture and designing basic wireframes, development of a functional prototype could proceed.

3.4.6. Information Architecture Outline

Information Architecture is the structural design of shared information environments (‘Information architecture’, 2015). For the website it includes appropriate organising and labelling of the individual pages in order to facilitate users navigation and interaction with the website.

When creating information architecture, several key principles have been followed (Krug, 2006):

- Using hierarchy depth proportional to the website size
- Creating persistent top navigation
- Using menu highlights for the currently active section of the site
- Graphical separation of the navigations from the rest of the content

- Using breadcrumbs for navigation
- Utilising blogposts categorisation

3.4.7. Wireframe Design

A website wireframe is an outline that represents the visual arrangement of a website. The wireframe depicts a page layout or arrangement of the website's content, including interface elements and navigational systems ('Website wireframe', 2015). As we didn't plan to modify the blog template layout significantly, we designed only wireframes for the title page and general web page. Because one of the requirements for website was responsive design, we created wireframes for the largest and the smallest screen with intention of readjusting large screen design for all other screen sizes in-between.

3.5. Stage 3 – Construction

The construction stage started at the end of June and lasted until mid-September. Internally it was divided with the first construction milestone in second half of July when the outer environment represented by skybox was created, textures were prepared and an initial layout of the scene set up. For the Conversion User Guide, a functional prototype of blog was working and graphics design was established before reaching the first milestone. In the second half of construction stage, all 3D modelling was done and models were placed into the application environment. With several iterations of refinement, the lighting was adjusted and virtual tour track set up. At the end of construction stage user testing of application was performed.

3.5.1. Functional Prototyping

Website development started with launching a blog based on the WordPress content management system under our domain. After choosing a convenient template that would approximately work with the wireframe designs for the rest of the website, we performed several adjustment to the layout via WordPress appearance options and modifying the WordPress template code. After setting up top navigation based on the information architecture, we readied the functional prototype.

3.5.2. Graphics Design and Style Creation

We chose a dark theme for the blog, which had created the base for the colour scheme. We sought accent colours for the menu items and links, but in similar hues. For generating the colour scheme we have used Adobe Kuler colour wheel online tool (Adobe Kuler, 2015). After settling for colours, we designed simple logo with the name of the project.

The style programming commenced with editing of CSS templates from the WordPress theme. Because the chosen theme already included responsive layout, we mirrored that in the CSS styles created for the rest of the site. Otherwise, we tried to keep the design spacious , clean, and with subtle motion graphics.

3.5.3. Template creation

Website templates containing code common to all pages facilitate the maintenance process once pages are created and adjustments are needed. We created two templates (Appendix F), one for the code above the content and one for the code following the content that are inserted into every page via a PHP call. To deal with custom menu highlighting and breadcrumb navigation, we designed PHP functions that operate with page's name and position queries.

3.5.4. Functionality programming

The first functionality that had to be developed was the moving carousel for the title page. For that purpose, we used the Bootstrap framework (Bootstrap, 2015) in combination with CSS to style carousel elements. We used different styles for each of the four layouts. After that we developed top navigation, so it would behave in the same way as the blog and on the rest of the website. We used JavaScript functions (Appendix G) for creating sliding behaviour of the top banner and change in navigation appearance for the smallest layout.

For the questionnaire functionality, we created simple HTML form with PHP function for validation and processing. We haven't linked the questionnaire page with the rest of the website in order to avoid spammers and bots. The questionnaire results were serialised with PHP function and posted to text files on server with timestamp, from where they were collected.

3.5.5. Camera calibration and initial layout

Camera calibration had an important role in setting up proportions for the outer world (skybox) and consequently for the rest of the scene. As previously mentioned (Section 2.2.1), for the camera calibration it is necessary to know the vertical viewing angle, the aspect of the viewing window given by ration of width and height, distance of a near clipping plane from centre of projection (camera position), and distance of a far clipping plane.

Setting up the near and far clipping planes is relatively straightforward, usually the near plane is very close to the camera and the far plane is behind the skybox, so the whole scene in distance is visible at all times. For our application, we found it convenient to set up the window size in a same ratio as that of the painting. At the end, the only unknown left was the vertical viewing angle. We needed to know it at a given point in time to pinpoint the skybox position and size from the painting correctly onto the one of the skybox image planes.

To estimate the right viewing angle, the linear perspective analysis performed earlier (Section 2.1.3) provided us clues with a distance point position on the horizon. A distance point that is further away from the vanishing point indicates a smaller vertical viewing angle than a distance point closer to the vanishing point (Figure 31). This calibration can be made even more precise later if the ground texture with regular pattern is available or geometric objects of the cuboid shape are to be positioned in space.

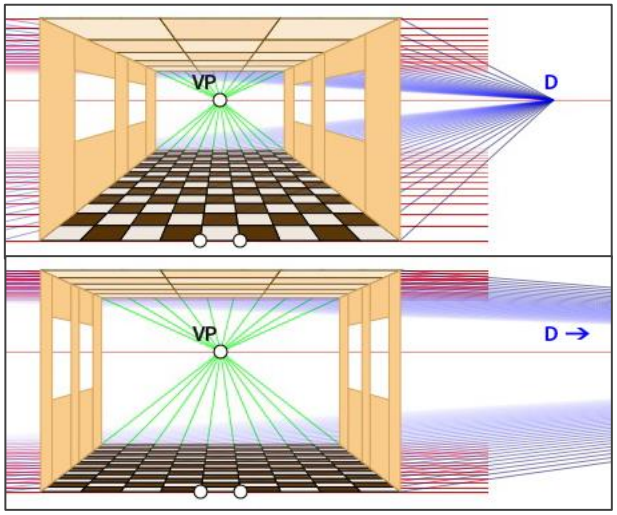


Figure 31: Wide vertical viewing angle with close distance point and narrow viewing angle with distance point far away

3.5.6. Painting and texturing

Painting and texturing activity varies in several different ways depending on the scene elements created. We distinguish simple texturing and painting with the uniform texture for the whole object, projective texturing with images projected onto parts of the object and a skybox texturing.

For simple texturing we needed to create homogenous textures, onto which our 3D models can be unwrapped. Sometimes, it was necessary to recreate a larger texture sheet from the smaller texture source. This was done in Adobe Photoshop with the clone stamp tool, or in more rigid way via seamless texture tiling. To create seamless edges of the texture tile, we have designed algorithm (Appendix H) that enable us to set size of overlay of the tile that is modified and provides smoothing functions at the end.

Projective texturing requires us to transform textures from the painting that are viewed at an oblique angle into an orthogonal view. As we showed in the Section 2.1.2, this can be done via matrix multiplication. To calculate the transformation matrix we need four correspondence points between the original and orthogonal plane, where no three points are collinear. Once the projective matrix *T* is known, Matlab Image Processing Toolbox provides functions for the projective transformation (Matlab, 2015):

```
t_proj = projective2d(T)                    (3.4)
```

```
I_projective = imwarp(I, t_proj)            (3.5)
```

Texturing a skybox is the most complex undertaking from all texturing tasks because of extent of newly generated texture. A particular texturing procedure strongly depends on the actual painting. With homogenous sky background and few objects (clouds, stars) placed over, the easiest way is to fill the image with the background colour and use the Adobe Photoshop clone stamp tool to create the objects from the source texture. Their size and shape can be

modified in non-destructive way with Photoshop liquify tool (Figure 32). A similar strategy can be used for a skybox with a vertical colour gradient background; however it is necessary to adjust horizontal lines over deformation zones caused by cube shape of skybox (Figure 33). Likewise for this adjustment, the Photoshop liquify tool can be used.

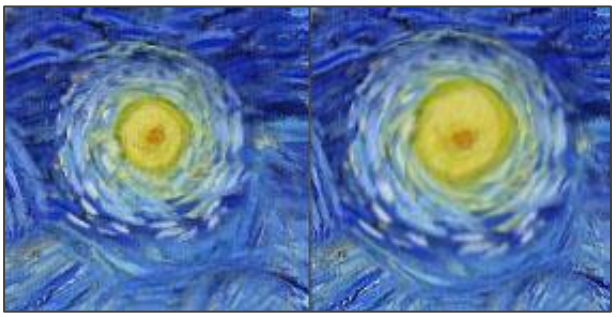


Figure 32: Changing size without damaging surrounding texture

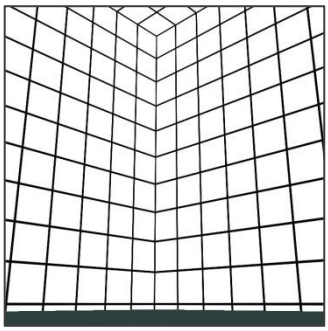


Figure 33: Skybox deformation around edges and corners

The most difficult skyboxes to create are those with very artistic textures that are not easily reproducible. For these, a good starting point is to reproduce an available texture in mirror-wise way around the whole perimeter of the skybox and to do overpainting with clone stamp tool, which simulates brush but its paint is a texture from defined source. The blending between foundation and overpaint depends on the quality of texture – if texture has lot of contrast, too much blending would destroy its original artistic properties.

3.5.7. Modelling and rigging

Approaches and procedures used in mesh modelling vary widely depending on the objects to be modelled and the modeller's personal preferences. However there are several common techniques employable in any workflow to speed up the entire modelling process. Next, we explain these techniques in conjunction with the Blender software.

The modelling process usually starts with editing simple geometric shapes or simple model templates (e.g. human figure, tree). The modeller can select the whole model or a group of vertices and perform translation, rotation and scaling operations. New geometry can be added via subdivision of the whole model or selected faces, via extruding faces and the additional vertices can be adjoined to the mesh individually or in the rings around the mesh perimeter. Along the process, meshes can be viewed as solid objects or as a wireframe structures.

Mesh modelling based on editing geometric primitives can create very advanced models, especially when it is done by an experienced modeller. However, there is a set of tools in Blender called modifiers that enable the creation of more optimised, extensible and easier to manipulate models (CG Cookie, 2015).

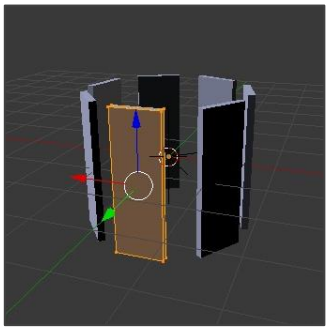


Figure 34: Array modifier creating multiple objects

The array modifier (Figure 34) creates instances of an individual object or a piece of an object. Each copy can be offset from the previous one in different ways (via reference objects) and adjacent vertices can be merged into single

geometry (Blender Foundation, 2015). This modifier is useful for the objects with regular polygonal floorplans and the same wall surfaces, for the objects consisting of repetitive parts or for repetitive objects that need to be arranged along a specific curve. This modifier enables us to create the whole object via one reference part.

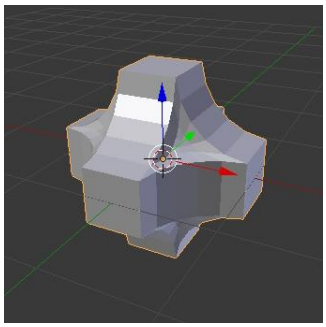


Figure 35: Bevel modifier applied on cube

The bevel modifier (Figure 35) enables us to slant vertices and edges of the objects with precise control how and where it is applied. It is possible to create a variable number of segments around the edge, set up a profile in which the edges should be slanted, or limit bevel intersection for the edges (Blender Foundation, 2015). This modifier is useful for modelling architecture, such as cupolas, arches, curved walls and roofs.

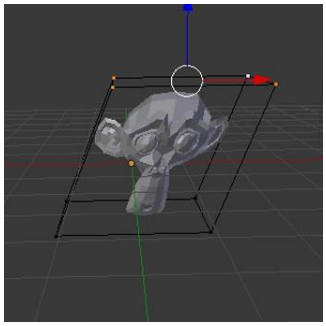


Figure 36: Deforming head with lattice modifier

A change in shape of an individual object can be done via the lattice modifier (Figure 36) without actually changing the object geometry. A lattice consists of a three-dimensional grid of vertices that are attached to the object and with their manipulation the shape of the object is transformed. This modifier is particularly useful for quick modification of the objects with a high number of vertices or to get a smooth deformation that would be hardly achievable via modelling geometry (Blender Foundation, 2015). It can be used for modelling architecture, but also to create variability in human figures and faces, or deformation of the trees.

There are many other useful modifiers in Blender and also tools for creating non-geometric amorphous surfaces on the top of geometric basis. This can be done via the sculpting brushes that provide modes for layering, flattening, pulling, creasing, nudging and many others.

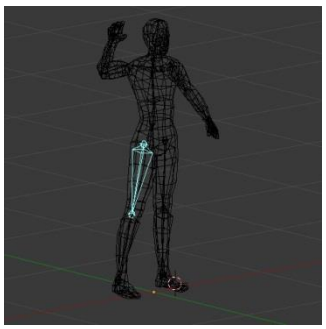


Figure 37: Rigging the character

The last specific type of modelling that can be potentially useful is rigging. Predominantly rigging is used in animation to move a character (Slabaugh, 2015), but in our case it can facilitate positioning of a human figure (Figure 37). Rigging basically means attaching digital skeleton to the meshes and by defining bones and joints it is possible to simulate realistic movement and arrangement of different body parts.

3.5.8. Implementation

The implementation discipline is, according to RUP, where “actual code is produced, subsystems integrated and the system implemented”(Shuja and Krebs, 2008, p. 131). In our application, integration of all subsystems was realised via the main Application class. Therefore between each two stages of development we kept version control for the Application class and newly designed class to record of the last working version. If the new

class required modification in shader programs, their last working versions were also checked-in before proceeding with further development. At the end of each stage of implementation the quality of the code was evaluated with the focus on correctness, low coupling, logical structuring and readability.

Part of the implementation consisted of the positioning and transformation of the 3D models created in the production pipeline. This was done via a MatrixStack class that enables one to the preservation and restoration of matrices in a similar way to a stack data type– the last pushed element is the first one to be popped and it is possible to access the top stack element. Since one of the 3D model properties that can be defined in Blender is material reflectance, by placing models into the virtual environment with global illumination the lighting and reflectance parameters need to be readjusted.

There were several additional classes that we needed to adjoin to the original template during implementation. The Track class had to be programmed in order to run a virtual tour around the painting environment. The trail for the camera motion was defined in the world coordinates by several points (Appendix J) and the rest of the route was created along the splines, which were interpolated between the points.

As we had decided to use a heightmap to model terrain in our environment, we had to add a class for terrain generation as well. The choice for a heightmap terrain as opposed to mesh terrain was due to performance reasons, since proper terrain modelling would require high-poly models and our polygon budget was limited. Besides the Heightmap class expansion and modification we had to program texture binding and blending in the fragment shader.

3.5.9. Testing

The testing discipline consists of component testing during the development when new classes were added, user interface testing, performance testing and the final user evaluation of the application via demonstration video and questionnaire.

Performance testing was done after the addition on individual meshes and for turning on and off various features. The frame rate indicator was observed for the individual objects (or group of related objects) and also in cumulative way. In case of unexpected burden, the models were readjusted and their extensive geometry simplified.

Since the user interface is very straightforward, simple functional testing was done at the end of the application development. The component testing for additional classes and shader programs modification was done before starting work on the new component. For some of the components, additional functions were used during development and testing and later removed.

The user evaluation was performed via the website publishing the video of the application and questionnaire (Appendix I). There weren't any special requirements for testers other that having laptop or desktop computer and internet connection.

3.6. Stage 4 – Transition

The last stage of the project, the transition, took place from the middle of September until the end of the project. During this stage, project outcomes were prepared for submission, content was uploaded on the project website and a promotion strategy for the project was formulated.

3.6.1. Deployment

The function of the last discipline - Deployment - is to "manage the activities associated with ensuring that the software product is available to its end users" (Shuja and Krebs, 2008, p. 157). For our application that involved final checking and additional commenting of the source code, creating a folder with executables and packing it into ZIP file. A simple user manual on how to navigate around the application was added to final package as well. Algorithms were packed together with make files and README files with explanations and instructions on how to run the algorithms. A video of the application running was created for external users and published on the project website.

3.6.2. Content Supply

The main content of the website contains the blog learning articles and individual project scene elements. Those were published in their respective sections and appropriately tagged. For each project scene element a display image and a ZIP file were created to make it possible for the users to browse and download.

3.6.3. Promotion

At the end of the project we located several 3D modelling and game object websites on which the products of different artists are displayed. In future we can publish several of our models on these sites to make our website known among the designer and developer community.

4. Results

This chapter describes the products to be created at the beginning of the project as well as other outcomes of the previously described methods. We start with preliminary research results, which were essential for advancement of the project and comprise a list of tools and software with their pros and cons of usage and specified recommendations for painting selection. Then we present results of the production pipeline, followed by results concerning the whole application. At the end we present the Conversion User Guide and the full project website.

4.1. Preliminary Research Results

4.1.1. Painting Selection Criteria

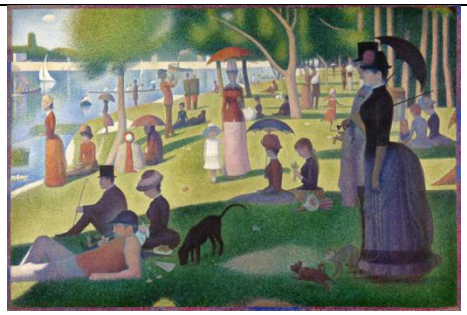
The criteria for painting selection were determined at the beginning of the project subsequent to completion of modelling courses and were refined throughout the project as more experience with conversion was acquired. Each criterion contains a description with indicators how to recognise it in the painting and example paintings that are hard and easy to convert from the point of view of that particular criterion.

Criterion	Discreteness of the elements	
Why is it important	To convert a painting into 3D scene, it is necessary to extract individual elements from the painting before creating 3D models. If object parts or object itself is “mixed” into the scene, segmentation is hard or impossible.	
Indicators	Element boundaries are visibly separable with contours, shading, different colours or textures. Overlay between two objects is very minimal and paint from one object is not blended into the other object.	
Difficult to convert	Claude Monet: Soleil Levant	

Easy to convert	Giorgio de Chirico: Love Song	
Criterion	Total count of the elements	
Why is it important	Each element close enough to the front of a scene must be represented by 3D mesh model. Even with low-poly mesh models, having 100+ different models would create potential performance problems for the application and would require large time budget.	
Indicators	Total number of individual elements separable from the rest of the scene (people, animals, buildings, trees) or for complex elements total number of their parts.	
Difficult to convert	Pieter Bruegel: The Procession to Calvary	
Easy to convert	Salvador Dali: The Persistence of Memory	
Criterion	Surface complexity of the elements	
Why is it important	When 3D conversion is done by modelling, objects with q large proportion of geometric shapes can be reproduced more faithfully than complex scene elements with amorphous surfaces. They also usually require high-poly mesh models.	

Indicators	A good indicator for surface complexity is material – amorphous surfaces are often made of rock, soil or cloth, but sometimes even complex objects with relatively geometric components can appear amorphous, such as trees or decorative objects from architecture.	
Difficult to convert	Sandro Botticelli: Temptations of Christ	
Easy to convert	Francesco di Giorgio Martini: Architectural Veduta	
Criterion	Texture complexity	
Why is it important	The conversion process requires a texture reconstruction and generation (from small source), which is easier to do for relatively uniform textures than for artistic textures. With artistic textures it is sometimes difficult to recognise an object's shape and components.	
Indicators	Distinctive brush strokes creating irregularities in painting's texture are the most obvious indicator. The lines in texture are almost as strong as those separating the individual scene elements and creating seamless tiles from the original texture would distort it distinctively.	
Difficult to convert	Vincent van Gogh: The Starry Night	
Easy to convert	Hieronymus Bosch: The Garden of Earthly Delights	

Criterion	Depicted reality	
Why is it important	When a painting reconstruction is done, many assumptions are made in order to recover missing depth and relative size information. When painting contains objects that are not present in everyday reality it is harder to estimate size and positioning of the objects.	
Indicators	The painting contains objects not known from the real world or real objects whose parts are deformed or grotesque.	
Difficult to convert	Salvador Dali: The Temptation of Saint Anthony	
Easy to convert	Karl Gustav Rodde: An Italian Village by a Lake	
Criterion	Scene structure	
Why is it important	Scene structure provides a framework for positioning the elements, an estimation of their sizes and a camera calibration. Without this information, the camera calibration and scaling will instead become a trial and error process that may not lead to consistent results.	
Indicators	Lack of straight lines in the scene elements, few identifiable scene layers containing objects discrete objects.	
Difficult to convert	Albert Bierstadt: Storm in the Rocky Mountains	

Easy to convert	Georges Seurat: A Sunday Afternoon on the Island of La Grande Jatte	
Criterion	Scene perspective	
Why is it important	Linear perspective provides important information about parallelism and orthogonality of the painting scene elements. It is possible to recover the original 3D structure and orientation of the painting's geometric objects via linear perspective analysis as well as object relative sizes.	
Indicators	It is possible to find one or two vanishing points, at which all parallel lines going towards horizon meet. Sometimes it is possible to find a distance point via traversal intersections.	
Difficult to convert	Ambrogio Lorenzetti: The Effects of Good Government on City Life	
Easy to convert	Raphael: The School of Athens	
Criterion	Objects volumetrics	
Why is it important	Object volumetrics have influence over the way light is bent from the surface of objects. If a scene contains flat objects or objects with a very small amount of shading it is impossible to recover any 3D shape. In cases where the shading that indicates an object's volume is inconsistent, it might be impossible to establish a global scene illumination.	
Indicators	Objects' textures lack shading, objects lack volume and a scene has very 2D look.	

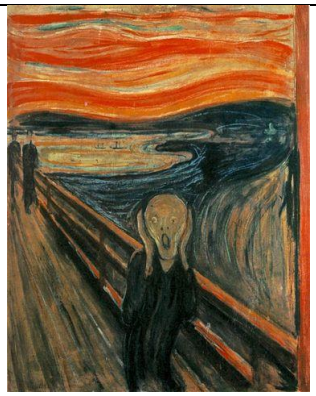
Difficult to convert	Joan Miro: The tilled field	
Easy to convert	Andrea Mantegna: The Agony in the Garden	
Criterion	Horizontal Viewing Angle	
Why is it important	Horizontal viewing angle establishes a proportion of panorama (360 degrees) that is visible from the whole scene. If the horizontal viewing angle is too narrow, only small proportion of the scene is visible and therefore reconstruction of the whole scene requires more understanding of the painting's aesthetics and implied reality.	
Indicators	A narrow horizontal angle is indicated by a painting's vertical scene dimension much larger than the horizontal scene dimension. With a given ratio of painting scene's width and height, narrow vertical viewing angle means also narrow horizontal viewing angle.	
Difficult to convert	Edvard Munch: The Scream	
Easy to convert	Hendrick van Steenwyck the Younger: The Courtyard of a Renaissance Palace	

Table 2: Criteria for painting selection

4.1.2. Software for Conversion

To perform painting conversion, different combinations of software tools can be used. If users want to exclusively use open source software, GIMP, OpenCV, Blender and OpenGL would be a sufficient combination to do the same conversion as has been done in this project. For commercial software, Matlab, Adobe Photoshop, Maya and Unity provide advanced tools to make many tasks simple.

The following table provides a short summary of the conversion tasks requiring different tools and software options for each particular task.

Task	Available software
Environment integration and programming	OpenGL – is a specification not an engine for virtual environment creation and therefore it requires either a considerable amount of initial programming or use of the existing templates. Several OpenGL features relevant to this project have been described in chapter 2.
	Unity – is a game engine that includes an interface for 3D environment creation. It supports a wide variety of asset formats and similarly as OpenGL provides shader programs, global illumination, and camera.
Image processing	Matlab – has a specialised Image Processing Toolbox with functions for image filtering, enhancement and segmentation. Matlab uses very simple tailored functions for each task.
	OpenCV – OpenCV covers the same areas of image processing as Matlab, however it requires more programming.
	Photoshop – offers GUI for filtering and manipulation of an image. Segmentation can be done in a similar way as in Matlab – via finding object masks or colour thresholding. Preview of enhancement filtering is available to user in the filtering process. Image management is possible via layers.
	GIMP – provides similar functionalities as Photoshop for segmentation and filtering. However, it has a smaller range of filters than Photoshop.
Image analysis	Matlab – provides functions for texture analysis and also for plotting image colour spaces in 2D and 3D.
	OpenCV – provides many library functions that can be combined in different algorithms to extract and visualise colour and textures.
	Photoshop – enables thresholding and manual colour sampling, although the method is relatively inefficient. The important role Photoshop plays is in linear perspective analysis where many transparent layers need to be applied on the original image.
	GIMP – supports colour sampling and layers in similar ways as Photoshop. Older versions didn't support CMYK profile, but in recent years a plugin for basic CMYK support was added.

	Saliency Toolbox – runs in the Matlab environment (but is not a standard Matlab package). It provides analysis of salient features of an image from the perspective of human viewpoint.
Painting and texturing	Photoshop – contains the most important tool for texture recreation – the clone stamp tool and also the liquify tool for further non-destructive texture deformation. Photoshop also provides simple perspective transforms but there is no rigorous way to obtain good results.
	GIMP – contains clone stamp tools as well but it provides less advanced options for further texture deformation and manipulation than Photoshop , such as the cage tool.
	Matlab – provides perspective transformation functions needed for orthogonal texture reconstruction.
	OpenCV – also provides perspective transformation functions via matrix multiplication.
Modelling and rigging	Blender – contains features for 3D modelling, material definition, object texturing, sculpting, rigging, and also a physics environment to simulate real world object behaviour (such as cloth draping). It also caters for wide range of import and export mesh formats. Blender can be run almost on any OS platform. Downside of Blender is its interface for several features that is non-intuitive and misinterpretation can cause significant loss of time to the developer.
	Maya – is the most popular commercial tool in 3D modelling industry. It is possible to create in it all assets as in Blender. Nowadays, the major advantage of Maya compared to Blender is character animation, but that feature is not relevant to this project. Maya can be run on several different OS platforms.
	3ds Max –another commercial 3D modelling tool. Again, the main difference compared to Blender and Maya is the interface rather than the assets one can produce. Unlike Maya and Blender 3ds Max is available only for the Windows platform.
Specialised 3D conversion	Make3D –research software that runs under Matlab. It creates a single mesh from an image and enables orthographic projection of a texture. The resulting mesh is in WRL format that can be further edited in standard 3D modelling tools.
	Gimpel3D – is a proprietary tool that was developed for 2D to 3D conversion. It provides multiple tools for 3D manipulation, auto alignment, depth modelling and texturing. Even though Gimpel3D is an advanced tool, the downside is deprecated support and incompatible model formats for further reuse.

Table 3: Software for conversion

4.1.3. Guidelines for Capturing Painting's Aesthetics

Even though it wasn't originally specified in the proposal, we found it necessary to define certain guidelines that would accompany the development of the scene part that is not visible in the original painting.

Aesthetics may be perceived as something very abstract, but there is information to be extracted from painting that can help to capture aesthetics of the missing scene invention.

Colours

There are several different colour systems that classify colours. In general, colour is defined by its hue (given by light wavelength), saturation (how pure colour is) and lightness (value or tone of colour). It may not be that useful trying to characterise paintings based on individual hues, but in many paintings a prevalence of warm or cool hues can be identified. Normally, saturation would be the most distinct noticeable characteristic. One can discover the colour palette of the painting using this algorithm.

Textures, Material and Surfaces

Texture defines an appearance or consistency of object surfaces. Many textures are primarily defined by materials. Textures can be smooth or coarse, shiny or matte, even or uneven, high-contrast or low-contrast. Surfaces can be hard or soft, materials can be natural or manufactured. Different classification may be applied; it is just important to take notice of this aspect of painting even though it is less obvious than colour.

Shapes

Shapes of objects in a painting can be geometric or amorphous, regular or irregular, round or edge. This is very general grouping that can be tailored for more concrete categories of objects. It is crucial to identify at least which shape features are prevalent in the painting's world.

Lines

Lines in painting are usually more subtle than shapes; however they may be put there to guide and navigate viewer's attention. Lines can have horizontal or vertical orientation, they can be straight or curved, and they can be parallel or orthogonal to each other.

Fill of space

Fill of space is one of the factors that should be kept very similar in invented part of the scene as they were in original painting scene. A density (or sparsity) of the objects is one the main aesthetic pillars in design and architecture.

Scale

Scale is defined by a relationship of an object size to another object size. In art, special significance is given to the relationship between an object's size and the human body size. This relationship can be based on real world measurements or can be completely out of place. If the scale in painting is completely unrealistic, it may be hard to recreate new objects for the rest of the scene.

Proportion

Proportion is defined as a relative size of parts of the whole. Proportion can be distorted to various extents and it is very prominent in several painting styles. Again this is one of features that is hard to invent for new objects in the recreated part of scene.

Symmetry

There can be different types of symmetry in a painting. Point symmetry is when every part has a matching part in the same distance from the central point. Line symmetry is a mirror-like reflection of an object or a part of the object. Radial symmetry is an arrangement of similar forms or features around the central point. These symmetries don't always need to be part of the invented scene; however there may be an indication of some global symmetrical arrangement in the painting.

Motive and Theme

This parameter encompasses many categories and classifications and can be very hard to pinpoint. A good approach is to try to find a few different keywords that describe the essence of the painting and use them as starting point when a brainstorming for missing scene invention is done.

4.2. Production Pipeline Results

The primary input to the production pipeline was a painting image. Our chosen painting was *The Ideal City* by Fra Carnevale (Figure 38).

Figure 38: Input image after clipping damaged edges

4.2.1. Image Analysis Results

The image had no visible noise, so an analysis was performed on the original image data. For the colour analysis we have created an OpenCV algorithm (Appendix D). The algorithm first reduces the number of colour groups via converting pixel colour to the closest defined colour and then counts all pixels in a given group, sorts groups by size and generates the final palette. The closest defined colour calculation is based on the user input (e.g. user can reduce colours by factor 2 with almost no difference to number of colour groups or by factor of tens or hundreds to only few colour groups). The main logic of the algorithm uses look-up tables to test each pixel's colour and convert it to the closest defined colour. The algorithm creates an image with reduced colours so the user can determine whether the colour reduction is acceptable.

After that a hash table is created, with colour code as the key and number of pixels in a particular group as the value, for quick access when pixels are counted according to groups. Then groups that don't fulfil user requirement for the minimum pixel amount threshold are filtered out and the rest are sorted in descending order and graphically represented in the final palette. The scale of the colour circles in the final palette is logarithmic, otherwise for most of images only few colours would be visible to user. The same palette is created against black and white background for better perception of certain colour groups. The colour palette of our painting is shown in the Figure 39.

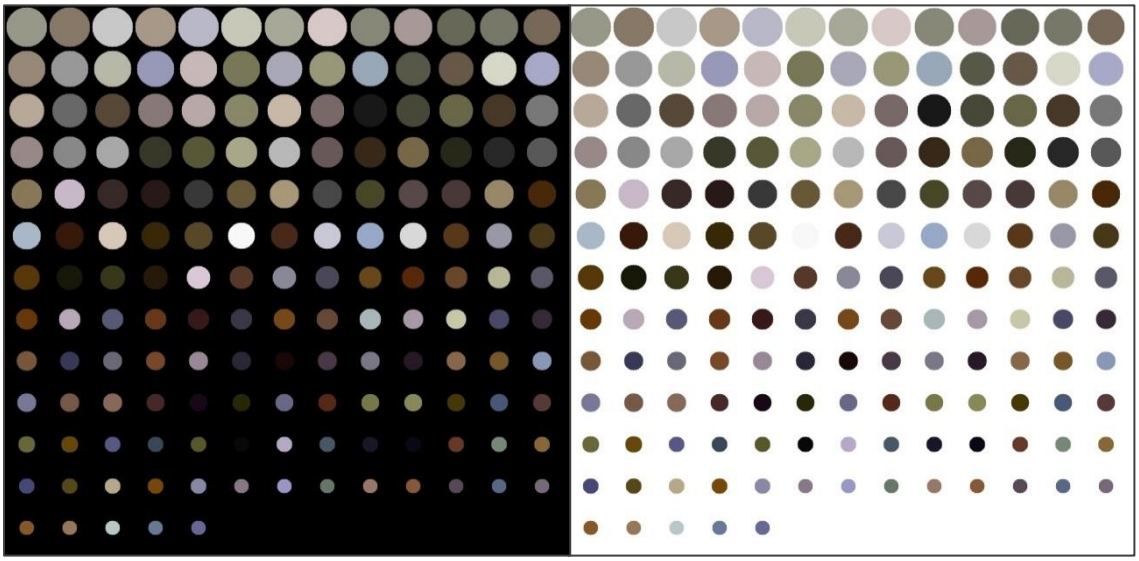


Figure 39: Colour palette of Ideal City painting (reduced by factor of 16)

The texture analysis has been done for particular textures individually. We have created the algorithm for seamless edge creation (Appendix H) for the square tiles that mix pixels from one edge into the opposite edge. The pixel positions from opposite edge that are part of the overlay are randomly generated with a higher density close to the edge and a gradual decline with increasing distance. This overlay, whose size is defined by the user, is then placed over the original pixels. Information about pixel contrast is provided to user before he chooses a smoothing filter (if any) for the overlay. If a filter is chosen, the algorithm creates images of smoothed overlay for the particular filter with different parameters (Figure 40). The user chooses whether he wants to apply any of these overlays to the final image.

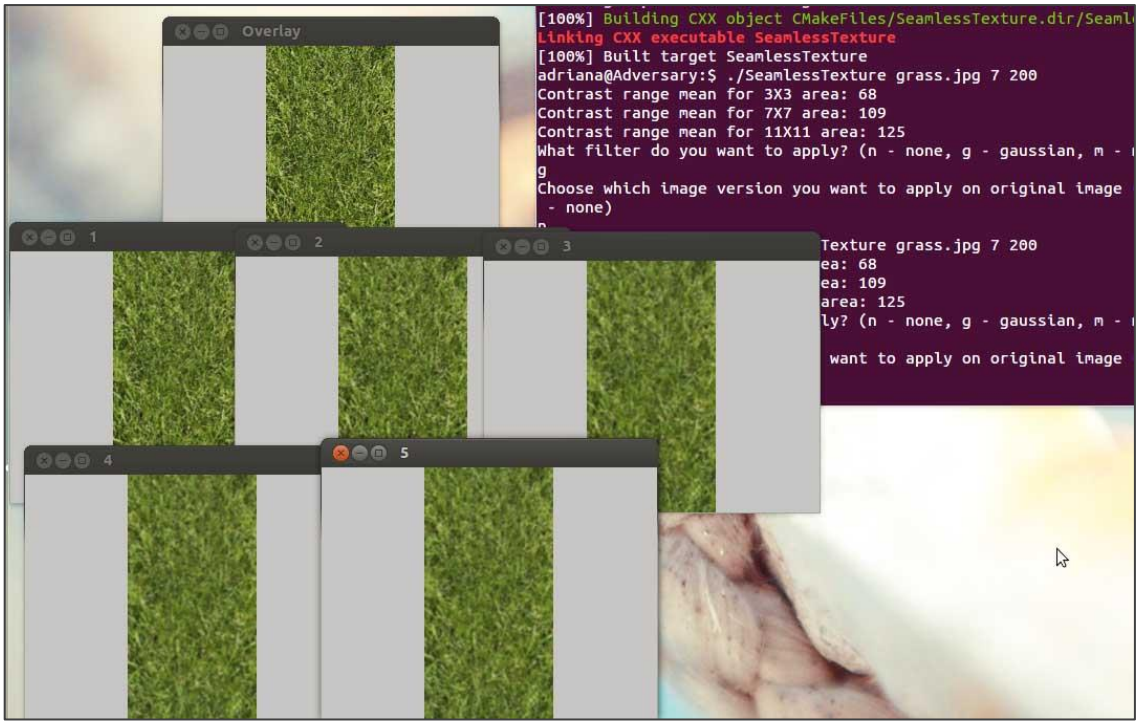


Figure 40: Different overlays created by smoothing filter

From the point of view of linear perspective analysis, the painting contains many important cues. There are parallel lines in both horizontal and vertical direction and there are regular geometric shapes (squares and octagonals). This enabled us to establish vanishing and distance points (Figure 41), important for the viewing angle estimation in camera calibration process. In this case, a distance point far away on horizon indicates quite a narrow viewing angle. The orthogonals and traversals split the painting ground into regular quadrilaterals, within which more precise positioning of the scene objects can be executed.

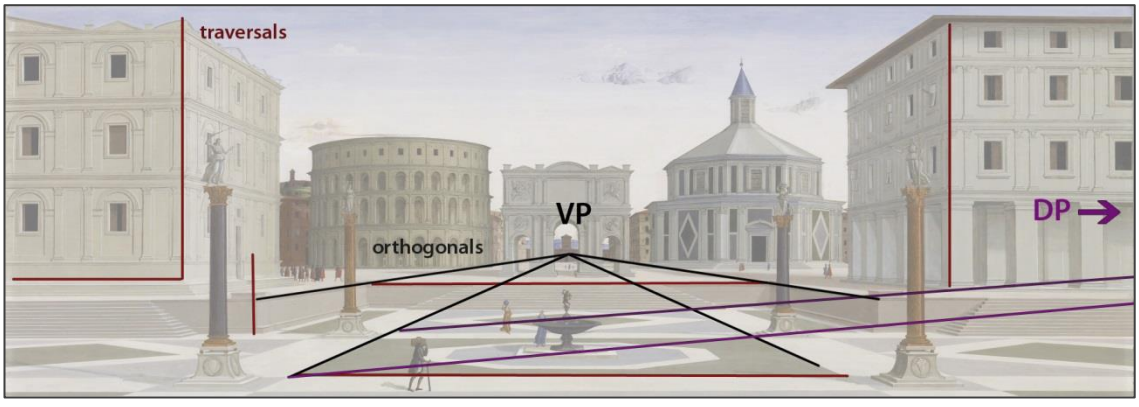


Figure 41: Linear perspective analysis

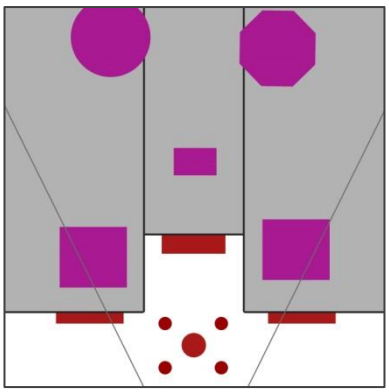


Figure 42: Layout map

Based on the linear perspective analysis we created an initial layout map (Figure 42) of the objects visible in the painting. The map is a view of the painting scene from the top. The positions of orthogonals and horizontal traversals were placed first to divide up the space. The map also contains the horizontal viewing angle of the observer and the main scene object positions. Distances between the objects were estimated based on horizontal traversal spacing given by the distance point. Because of high traversal density on the ground far away from the observer, the accuracy of distance estimation of farther objects is diminished.

The following results are from salient feature analysis that produced a step-by-step attention areas location (Figure 42) and a saliency map (Figure 43). In the Saliency Toolbox, we set up weights for all analysable features (colour, intensities and orientations) in the painting image to the same figure. The results of salient analysis gave us guidelines for modelling focus and the track creation.

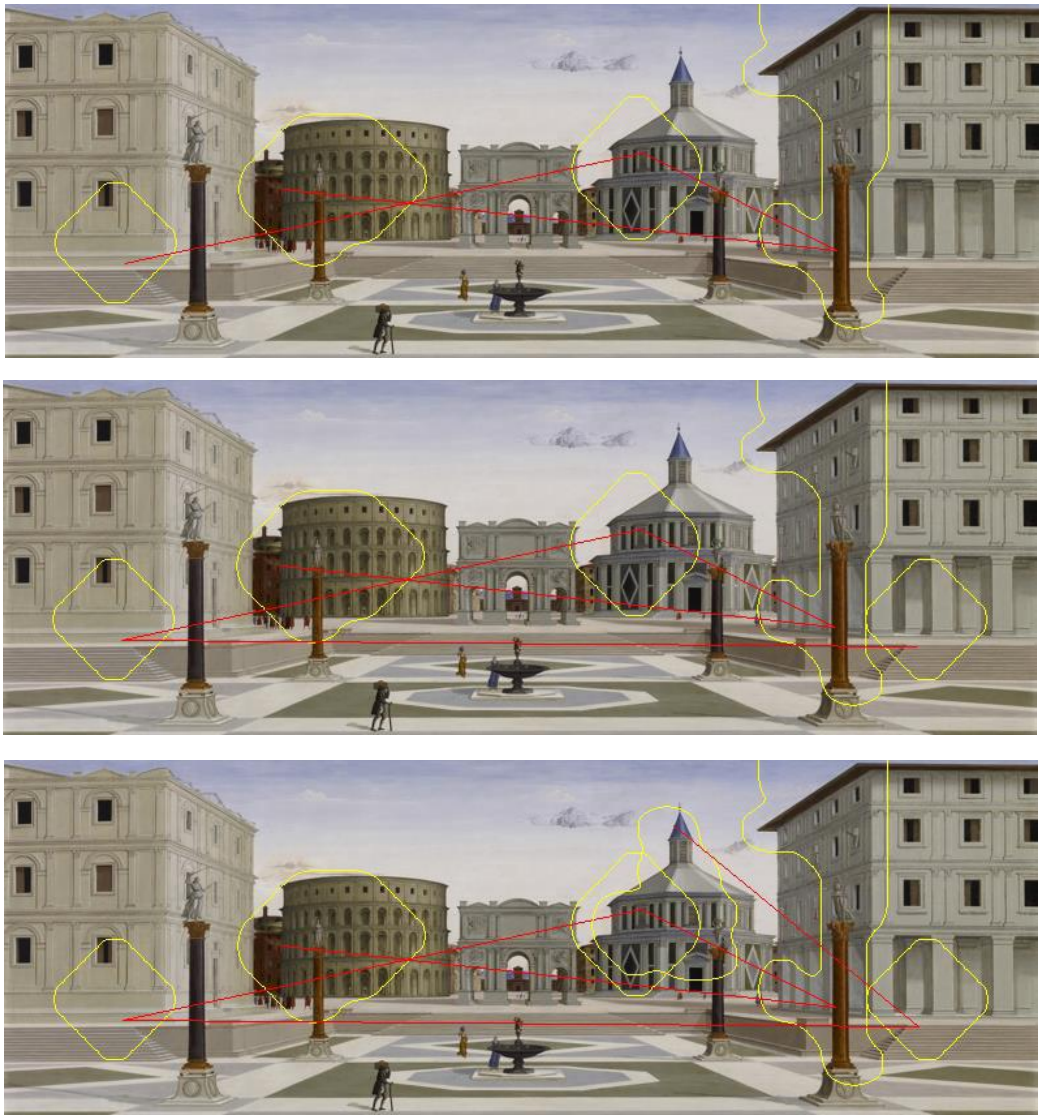


Figure 43: Attention area location

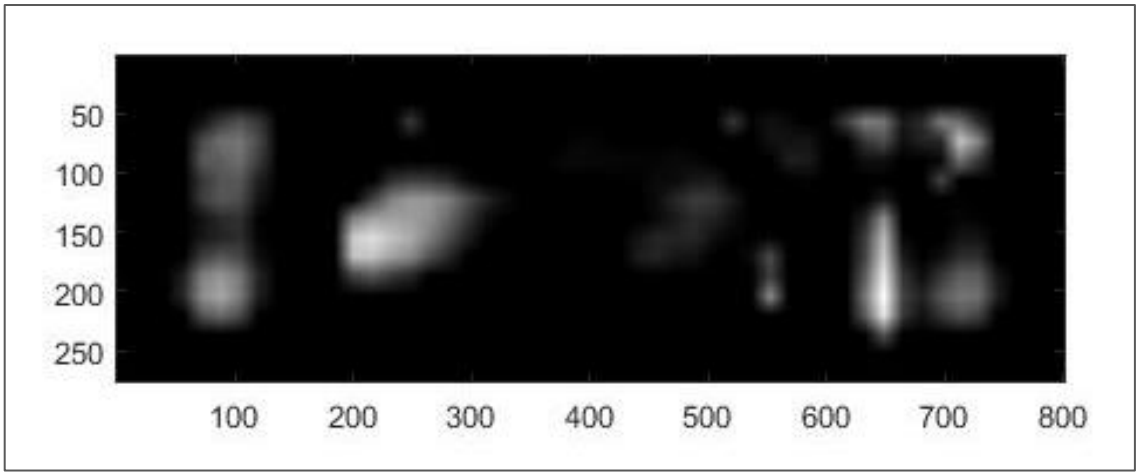


Figure 44: Saliency map

At the end of the image analysis we created a moodboard (Figure 44) to help us understand painting aesthetics for the missing scene reconstruction. Based on the guidelines for capturing aesthetics (Section 4.1.3), we identified that colour scheme is mainly muted (low saturation) with very small brighter highlights, with the prevalence of cool colours and lightness. Textures of the objects are very smooth without any small patterns. The only present patterns are large and geometrical with clean edges. The shapes of objects are predominantly regular and geometric. There are many both horizontal and vertical lines present. Painting contains lot of empty space and the scene objects are quite sparse. The scale of objects in comparison with human figures is on the large side. The proportions of all objects are normal without highlighting any particular feature. There is lot of line symmetry and radial symmetry both in textures, objects and positioning. The keywords that characterise painting are ideal, Roman, symmetric, architectural.

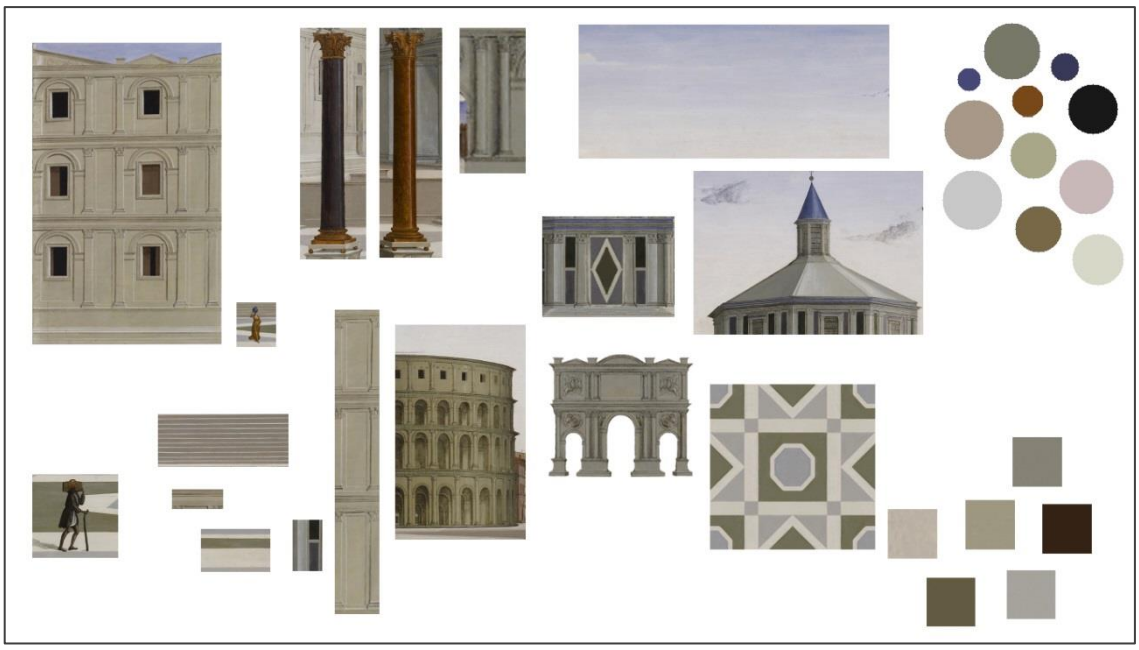


Figure 45: Painting moodboard

4.2.2. Skybox

The task of creating the skybox (Figure 46) based on the Ideal City painting was of medium difficulty. The sky texture didn't have obvious pattern but it contained a vertical colour gradient that is problematic from the point of view of deformation zones (as described in Section 3.5.6 and illustrated in Figure 33). Because of the painting ratio having much larger horizontal

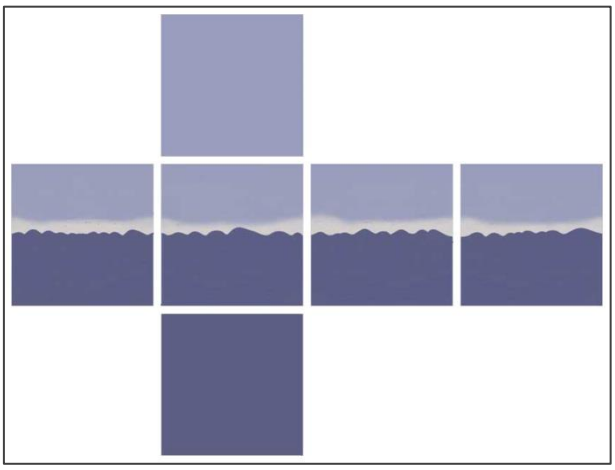


Figure 46: Ideal City Skybox

measurement than the vertical measurement, the gradient part from painting constructs only small proportion of skybox image height. We kept the rest of the sky the same colour as was at the top edge of the painting. The areas of the colour gradient close to the edges are deformed upwards to compensate the cube deformation zones.

Figure 47: Horizon line

There wasn't much information available about the horizon line except small image fragment in the middle of painting (Figure 47). Therefore we created the uniform single hue hill line based on this pattern for the whole skybox. The vertical position of the horizon line is arbitrary – it depends of the position of the terrain plane in the application. For our application, we had to position the terrain plane in the middle of the skybox cube height. The resulting effect of skybox modelling in the application is illustrated in Figure 48.

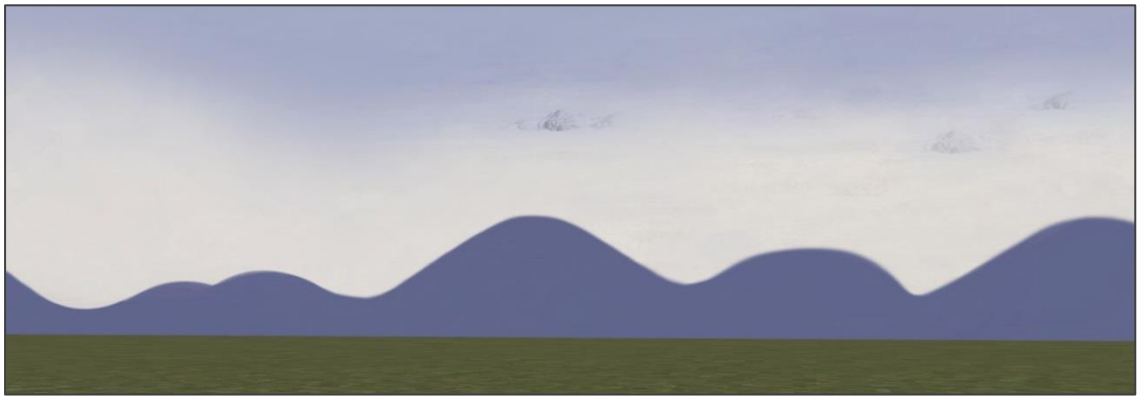


Figure 48: Skybox in application

4.2.3. Terrain

A visible terrain in the ideal city painting is mainly flat with the large tile pattern. There are building blocks positioned on the top of the tiled floor, but these were modelled as meshes. For the tiled floor a seamless tile texture (Figure 49) was created. There was an indication of the hill terrain in distance in the top left quadrant of the painting, but in the rest of the view the potential hills were covered by foreground objects. We have modelled the hill terrain around the perimeter of the city centre via heightmap (Figure 50) with one big hill that is also visible in the painting. Heightmap terrain is often used in virtual environments to avoid too large uncovered space where the planar terrain touches the skybox and to give an environment more realism. That was the reason why we chose to model the surrounding terrain this way. The resulting effect of terrain modelling in the application is illustrated in Figure 51.

Figure 49: Terrain tile

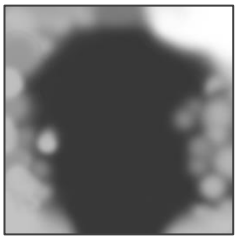


Figure 50: Terrain heightmap

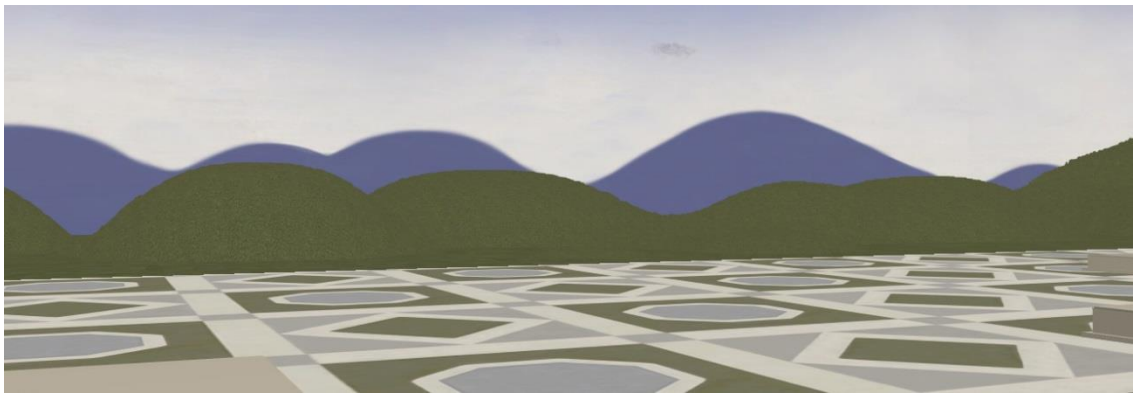


Figure 51: Terrain in application

4.2.4. Building Mesh Models

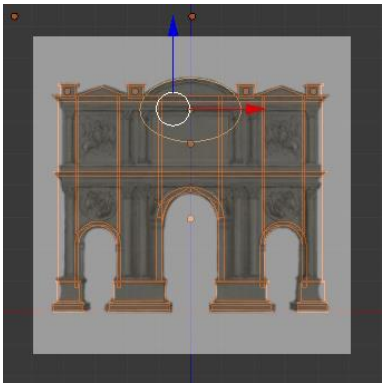


Figure 52: Orthographic projection of structure onto background image

The buildings in the painting are of several different shapes and complexity and therefore meshes created for each of them were built using different modifiers. There were five main buildings on which the modelling was focused: a colosseum, a triumph arch, octagonal coloured building, a pale yellow building (front left) and a grey building (front right). The rest of the buildings in background were modelled as simple textured cubes with roof.

The main building constructions were created in orthographic projection mode with vertices and edges projected onto the background image (Figure 52) of the building, so that the buildings' proportions were preserved. We tried to keep polygon count to the minimum so we modelled most of the facade geometry via texture projection. However all obvious overhangs and protruding parts were modelled as 3D. We used the array modifier for modelling bottom columns of the grey building, the mirror modifier for modelling the triumph arch and the boolean modifier[8] to model door entrance on the yellow building. The final vertices and polygon count for our main buildings mesh models is shown in Table 4.

Building	Vertices count	Polygon count
Triumph arch	1,155	1,063
Colosseum	416	203
Octagonal building	755	701
Grey building	1,688	1,187
Yellow building	818	638

Table 4: Vertices and polygon count for main buildings

[8] Boolean modifier enables a creation of new geometry at two meshes intersection.

The resulting building mesh models used in the application are illustrated in Figure 53.

Figure 53: Main buildings mesh models

4.2.5. Human Figure Mesh Models

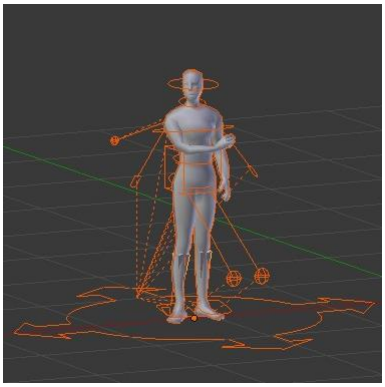


Figure 54: Figure positioning

There are several different human figures mesh models used in the application. They were modelled from single base human figure low polygon model [9] for which we have created a rig (skeleton) and positioned it to required pose (Figure 54).

Positioning the character was a prerequisite for any fitted clothes modelling. The fitted clothes model was created as a copy of the figure's body and resized so it would sit over the top of the body (Figure 55). The flowing clothes such as skirts or capes were modelled via physics simulation for the cloth material and sculpted to final shape. The human characters were painted with sampled colours at the end of the process. The statute figures were given single colour paint.

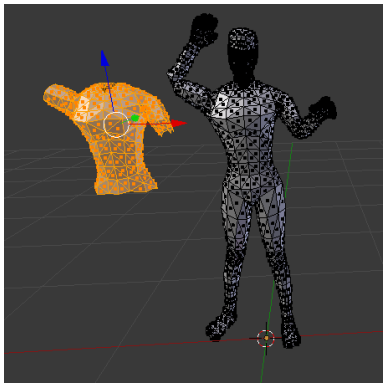


Figure 55: Fitted clothes modelling

The final vertices and polygon count for human figure mesh models is shown in Table 5.

Figure	Vertices count	Polygon count
Character1	11,495	10,313
Character2	5,179	4,771
Character3	6,768	6,008
Statue1	3,621	2,994
Statue2	3,673	3,030
Statue3	3,780	3,153
Statue4	4,982	5,216

Table 5: Vertices and polygon count for human figures

[9] Downloaded from http://www.blendswap.com/blends/view/17561 (Model credits: blender_player)

The resulting character mesh models used in the application are illustrated in Figure 56.

Figure 56: Character mesh models

The resulting statue mesh models used in the application are illustrated in Figure 57.

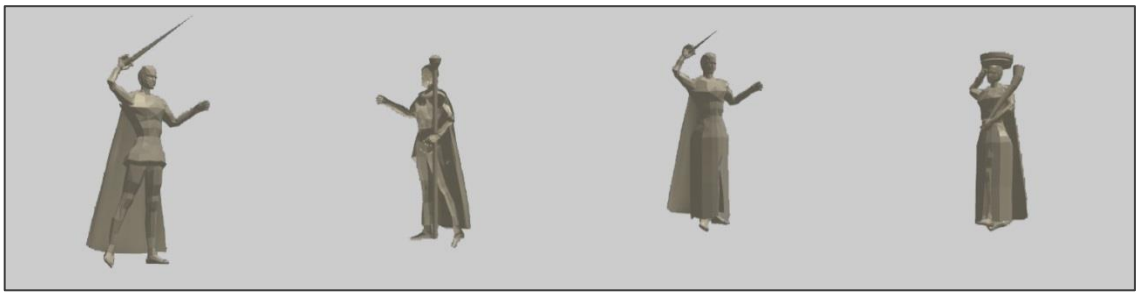


Figure 57: Statue mesh models

4.2.6. Other Mesh Models

The other mesh models in the scene include the prominent columns and a fountain, and other architectural pieces such as pavement blocks and stairs with a simple structure and very low polygon count. The columns and fountain were modelled in similar way to the buildings against the background image. The decorative top of the column was taken from the other column model[10] and its geometry was decimated to decrease polygon count for our resulting column. For curve creation in the stands of the columns a bevel modifier was used. Similarly for the model of the fountain, the angel on the top was modified from downloaded model[11] and the curves in model were bevelled.

The final vertices and polygon count for our other mesh models is shown in Table 6.

Object	Vertices count	Polygon count
Fountain	9,820	3,914
Column (4x)	18,369	26,693

Table 6: Vertices and polygon count for other main objects

[10] Downloaded from http://www.sharecg.com/v/68677/gallery/5/3d-model/corinthian-column (Model credits: Boardman)
[11] Downloaded from http://archive3d.net/?a=download&id=64f96144 (Model credits: Tom Rig)

The other mesh models used in the application are illustrated in Figure 58.

Figure 58: Other mesh models

4.2.7. Scene Completion

For the scene completion we tried to keep modelling within the aesthetics boundaries found during the image analysis. The main part of the completion was needed for the scene behind an observer. A Roman temple model was downloaded[12], recoloured and placed in multiple instances in symmetrical way behind the observer. The rest of the scene was filled mainly with heightmap terrain, reused pavement blocks, stairs and background buildings.

The polygon count for the temple mesh model in the scene is shown in Table 7.

Object	Vertices count	Polygon count
Temple (3x)	10,433	17.808

Table 7: Vertices and polygon count for temple mesh model

The scene completion behind the observer is shown in Figure 59.

Figure 59: Scene completion behind the observer

[12] Downloaded from http://tf3dm.com/3d-model/roman-ionic-temple-17272.html (Model credits: silviuq12)

4.3. Application Development Results

For our application programming and testing we have used a machine with the following specification:

- Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz 2.30 GHz
- 4.00GB RAM memory
- Intel(R) HD Graphics 4400 GPU with 1696MB of graphics memory
- Windows 7 Professional (64-bit)

4.3.1. Analysis and Design Results

In the Analysis and Design discipline, we have performed polygon budget estimation via setting up variations of rendering context and environment and testing it with different polygon loads. Table 8 shows the performance results (in frame rates per second) for the application with all lighting calculated in the vertex shader. Table 9 shows the performance results for the application with all lighting calculated in the fragment shader. In this first round of testing the impact of the individual feature was assessed. From the tables it is obvious that for high polygon load, an application with fragment shader calculations performs better than the one with vertex shader calculations.

Lighting computed in vertex shader				
Baseline (plane)				750 – 800 FPS
Object polygon count		**Plane + 5k**	**Plane + 50k**	**Plane + 500k**
No features	-	580 – 650 FPS	390 – 420 FPS	95 -100 FPS
Multi-sample anti-aliasing	2 fragments	330 – 370 FPS	240 – 280 FPS	84 – 94 FPS
	4 fragments	310 – 350 FPS	230 – 270 FPS	82 – 92 FPS
	8 fragments	190 – 220 FPS	160 – 200 FPS	70 – 82 FPS
Anisotropy filtering	On plane	390 – 440 FPS	310 – 340 FPS	88 – 95 FPS
Multi-pass rendering	2	370 – 410 FPS	230 – 260 FPS	50 – 53 FPS
	3	260 – 290 FPS	165 – 180 FPS	34 – 37 FPS

Table 8: Application performance with individual features when lighting is computed in the vertex shader

Lighting computed in fragment shader				
Baseline (plane)				700 – 750 FPS
Object polygon count		**Plane + 5k**	**Plane + 50k**	**Plane + 500k**
No features	-	550 – 650 FPS	420 – 480 FPS	108 -114 FPS
Multi-sample	2 fragments	330 – 380 FPS	240 – 270 FPS	90 – 100 FPS

anti-aliasing	4 fragments	330 – 380 FPS	240 – 270 FPS	90 – 100 FPS
	8 fragments	185 – 210 FPS	150 – 170 FPS	72 – 84 FPS
Anisotropy filtering	On plane	380 – 430 FPS	290 – 330 FPS	99 -103 FPS
Multi-pass rendering	2	360 – 390 FPS	235 – 265 FPS	57 – 60 FPS
	3	260 – 290 FPS	165 – 185 FPS	40 – 41 FPS

Table 9: Application performance with individual features when lighting is computed in the fragment shader

In the second round of testing, certain combinations of features were assessed together. Here again the application with fragment shader calculations (Table 11) performs on heavy loads better than the vertex shader one (Table 10).

Lighting computed in vertex shader			
Object polygon count	**Plane + 150k**	**Plane + 250k**	**Plane + 350k**
MSAA 4 + Aniso	153 – 165 FPS	115 – 124 FPS	95 – 101 FPS
MSAA 4 + Aniso + Multi-pass 2	85 – 88 FPS	64 – 68 FPS	54 – 57 FPS

Table 10: Application performance with combined features when lighting is computed in the vertex shader

Lighting computed in fragment shader			
Object polygon count	**Plane + 150k**	**Plane + 250k**	**Plane + 350k**
MSAA 4 + Aniso	155 – 170 FPS	125 – 135 FPS	103 – 112 FPS
MSAA 4 + Aniso + Multi-pass 2	85 – 90 FPS	68 – 73 FPS	56 – 60 FPS

Table 11: Application performance with combined features when lighting is computed in the fragment shader

4.3.2. Implementation Results

In the implementation discipline all assets of the scene were integrated together. The positioning was done first for the large objects and then the smaller objects of the scene followed. The light source was set up for the global scene in the world coordinates to match the shading of the objects in painting. A specular part of light was reduced to one third and all parts of light were assigned a neutral colour.

The whole scene assembly is shown from several different angles in Figure 60.

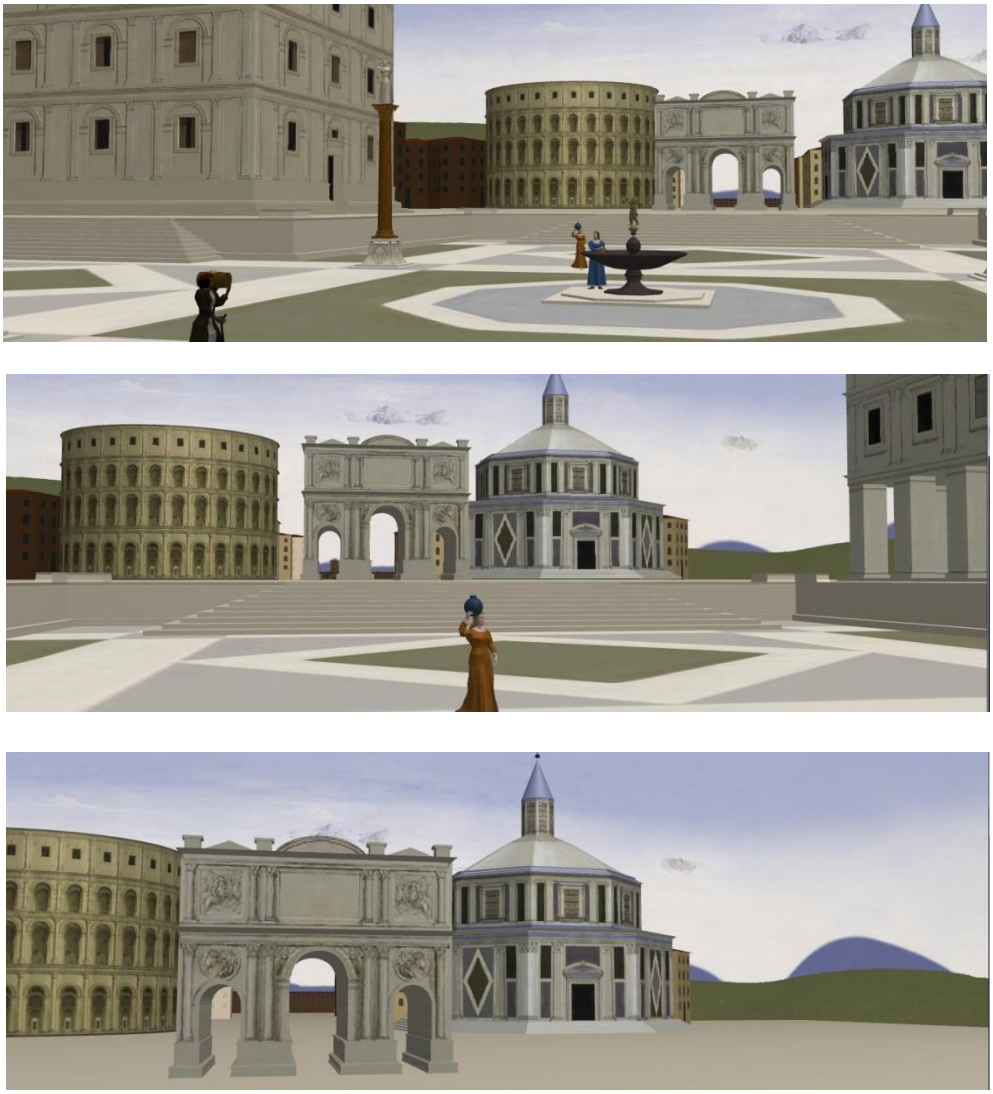


Figure 60: Painting walkthrough (screenshots from the application)

With scene assets positioned, the last element programmed was the virtual tour track. The control points for the track were set up (Appendix J) and the final track was interpolated between them. The top view of the track if it was rendered is shown in Figure 61.

Figure 61: Virtual tour track top view

Based on track points' position, different camera modes in the course of virtual tour were set up for the user to select. In front view, the camera viewing direction is towards the next point on the track. In side view, the camera is looking into direction that is perpendicular to its motion vector. In the top view, camera is positioned high above the terrain and is looking directly down. Then there is also free movement mode available to user to walk around the environment in his way.

4.3.3. Testing Results

The testing of the application was done via an internet questionnaire where the video of the painting walkthrough was shown under the painting image. Twenty people participated in testing. The results are as follows:

1. *Considering the video and painting, do you think the colours of the scene in the video in comparison with painting are:*

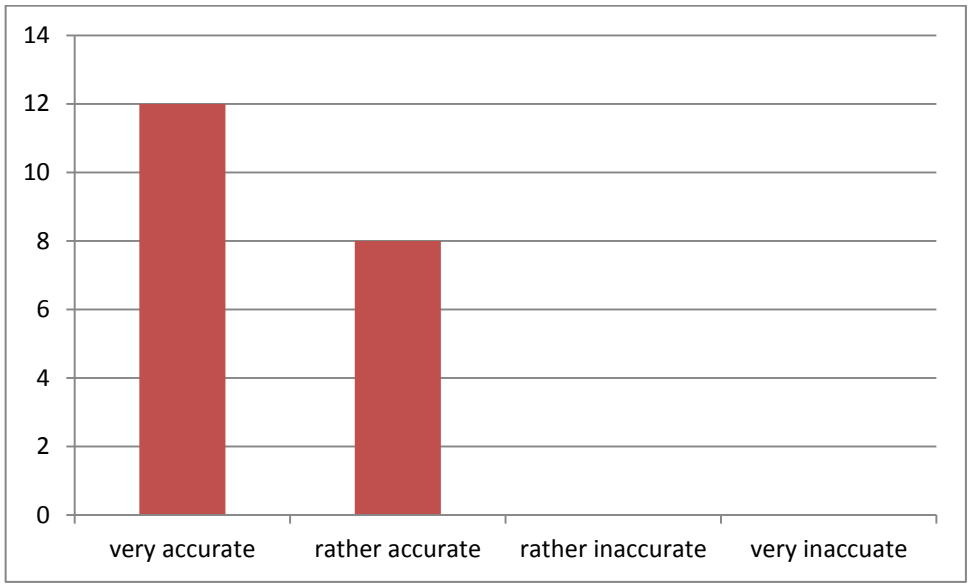


Figure 62: Question 1

2. *Considering the video and painting, do you think layout of the objects in the scene in the video in comparison with painting is:*

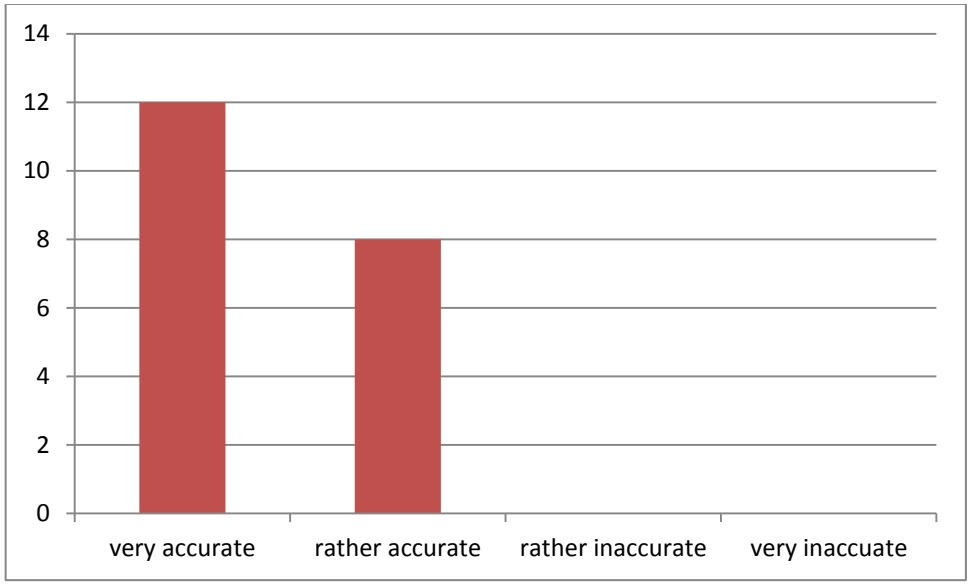


Figure 63: Question 2

3. *When you watch the scene on video, do you perceive the environment that is not visible on painting as:*

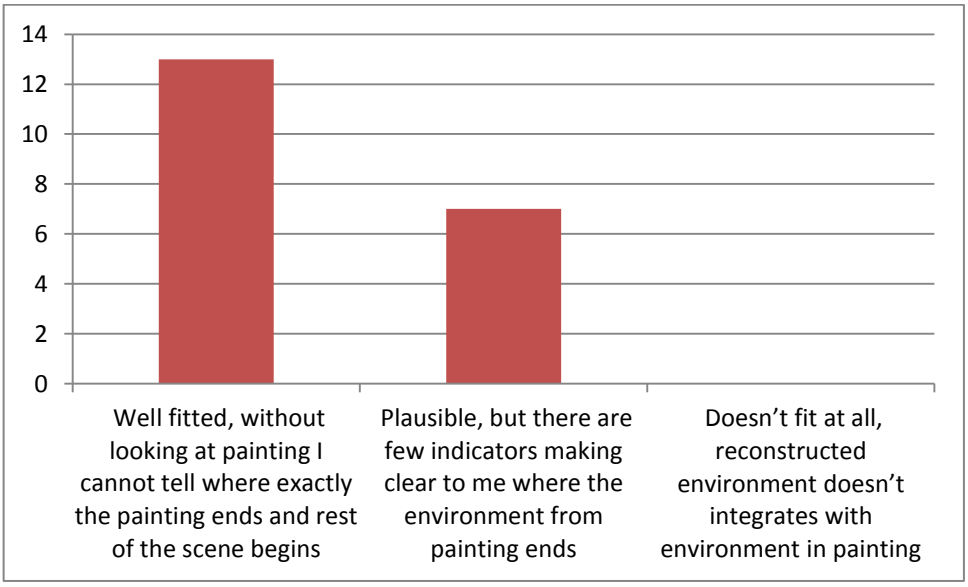


Figure 64: Question 3

4. *Considering the video and painting, do you think the shapes of objects (e.g. buildings, people, trees) in the video in comparison with painting are:*

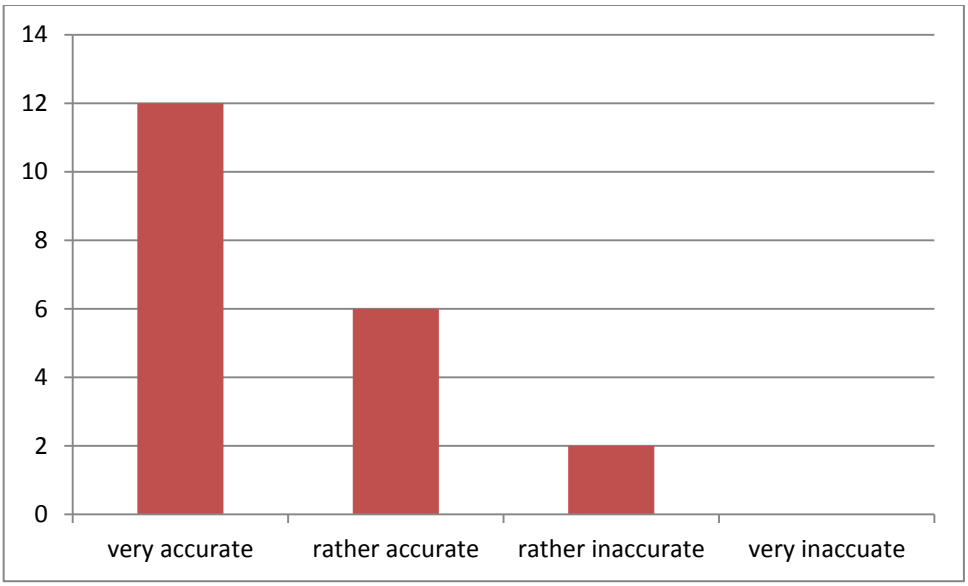


Figure 65: Question 4

5. *Considering the video and painting, do you think the sizes of objects (e.g. building, people, trees) in the video in comparison with painting are:*

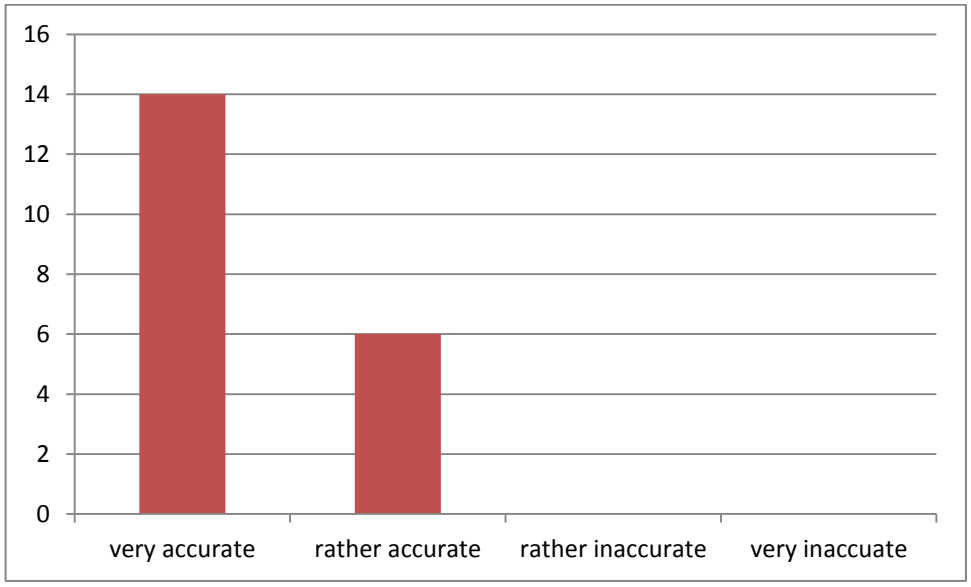


Figure 66: Question 5

6. *Would you be interested to have virtual tours around more paintings?*

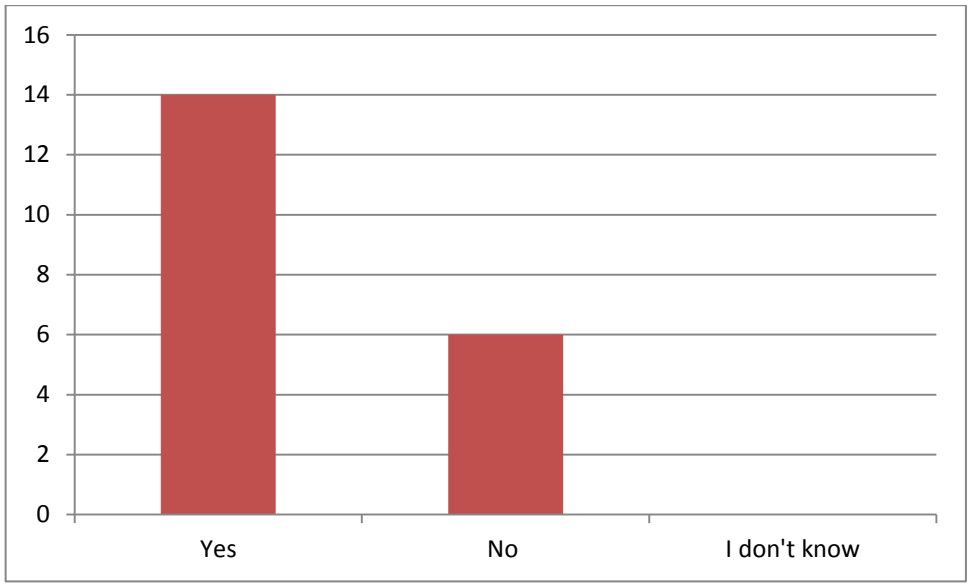


Figure 67: Question 6

Please tell us, if you have a preference for any painting.

The following preferences were stated:

- HR Giger

- The Entrance to the Grand Canal, looking East, with Santa Maria della Salute (1744)
- Gallifrey falls no more
- Salvador Dali
- The Night Watch – Rembrandt
- The Long Leg - Edward Hopper
- Any painting of Hieronymus Bosch
- One of the paintings of Venice by Canaletto
- Something with nature, or countryside

7. *Would you find it useful to have a guide telling you information about painting, when you having virtual tour?*

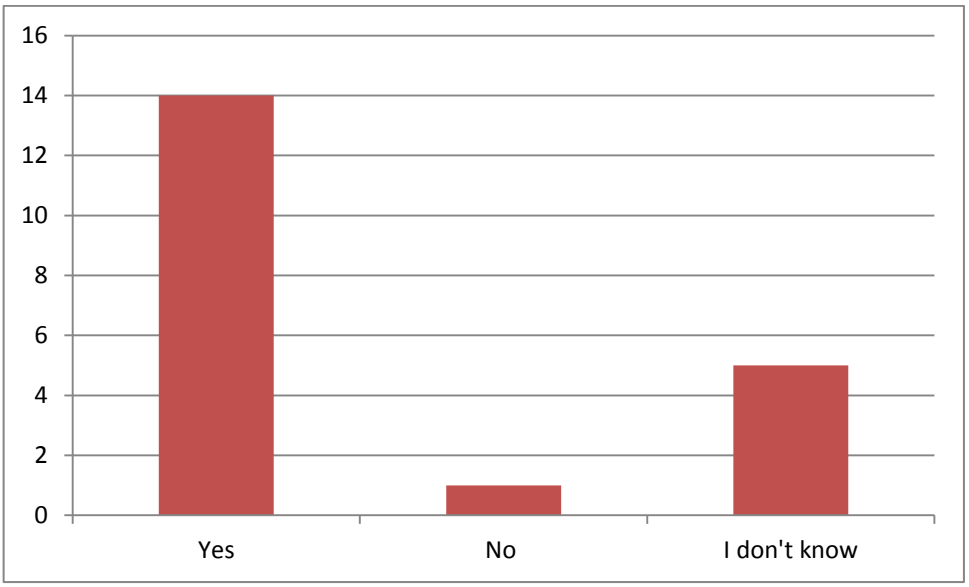


Figure 68: Question 7

8. *Would you like to play a computer game in the environment designed according to painting?*

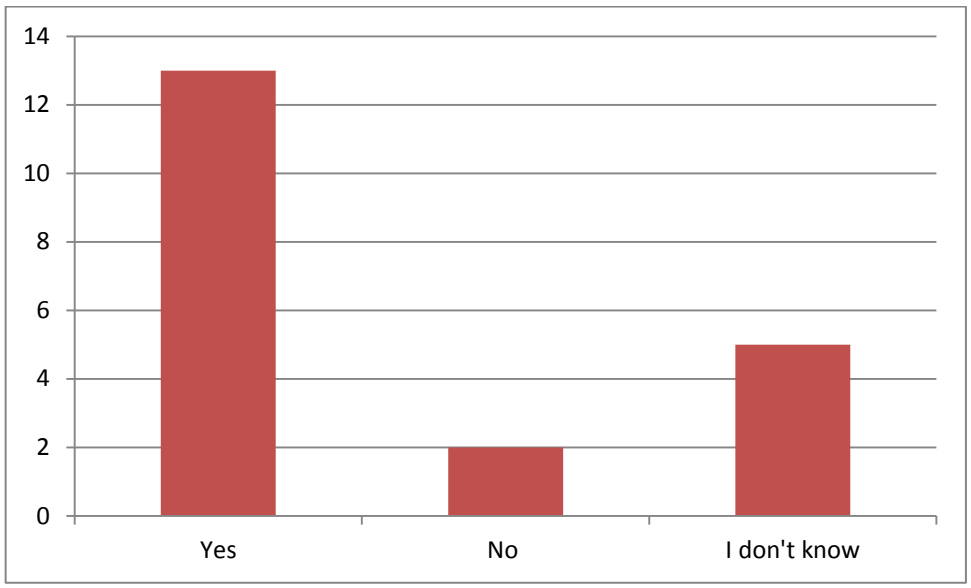


Figure 69: Question 8

9. *Your age group is:*

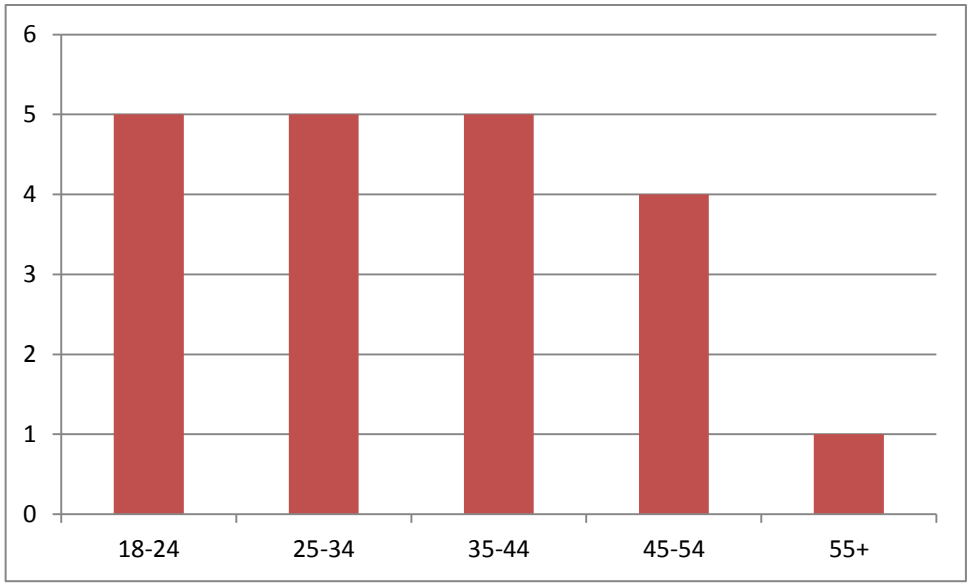


Figure 70: Question 9

Besides the questionnaire testing, performance testing was done for the application at the end. Our frame rate for presented was between 53 and 58 FPS. Based on this indicator we have decided to not implement shadows in the scene because it would require multi-pass rendering that would make the frame rate too low and would disturb the whole virtual tour experience.

4.4. User Guide and Web Development Results

The project website is available online at the time of project submission. The address of the website is www.virtualworld.sk/project and the Conversion User Guide is situated in the blog section.

4.4.1. Information Architecture

The Information architecture (Table 12) created for the website is quite simple, with 2-level depth and with different posts categorisation for the blog.

Information Architecture	
Home	
Blog	
	Categories
	Recent posts
Textures	
	Skyboxes
	Tiles
	Heightmaps
Models	
	Buildings
	Figures
	Others
Algorithms	
	Algorithm1
	Algorithm2
	Algorithm3
About	
	Project proposal
	Outcomes

Table 12: Information architecture

4.4.2. Wireframe Design

The wireframe designs were sketched for both home page (Figure 71) and other pages (Figure 72). The sizes of screens used in wireframes were the largest medium screen and the smallest medium screen. The intermediate screen sizes would have layouts similar to the largest screen.

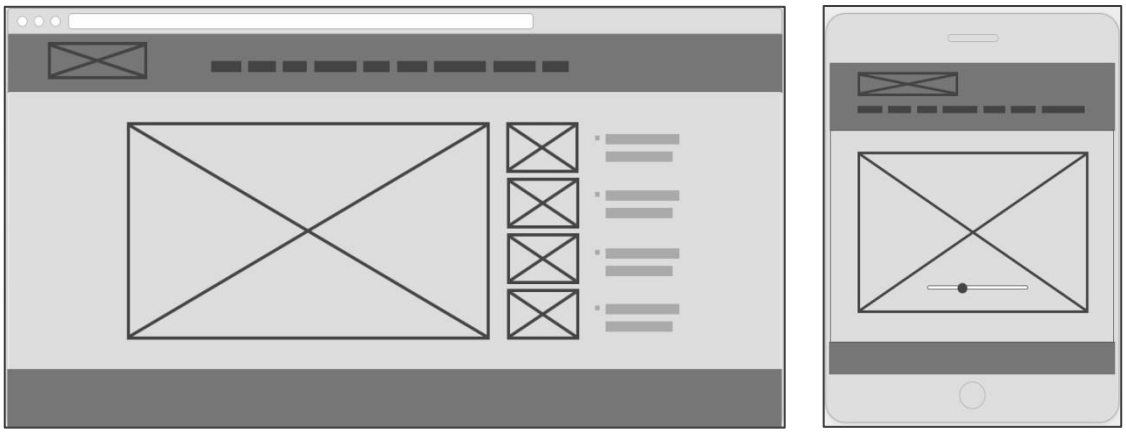


Figure 71: Wireframe design for home page

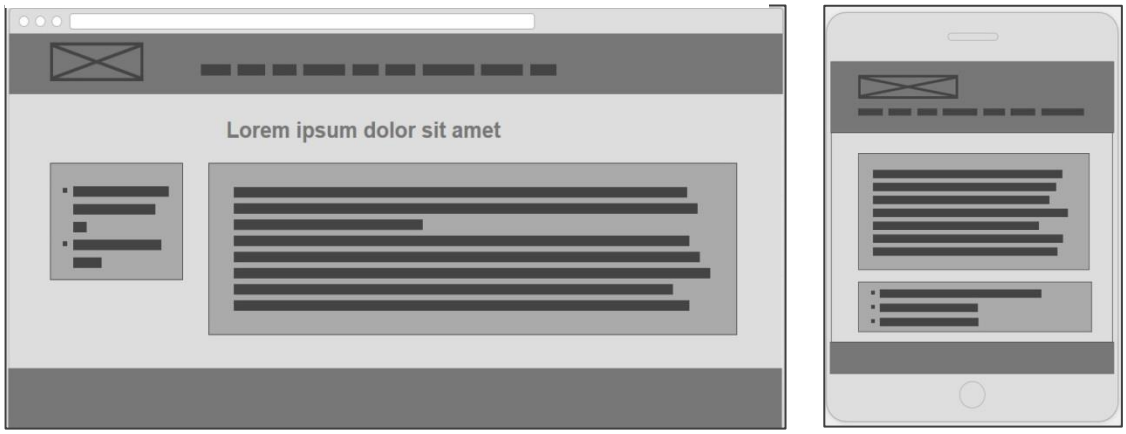


Figure 72: Wireframe design for general page

4.4.3. Website and Conversion User Guide Blog

The website was designed to be device responsive with four different layouts for various screen sizes. The home page (Figure 73) contains a sliding banner that advertises interesting links to different sections of the site.

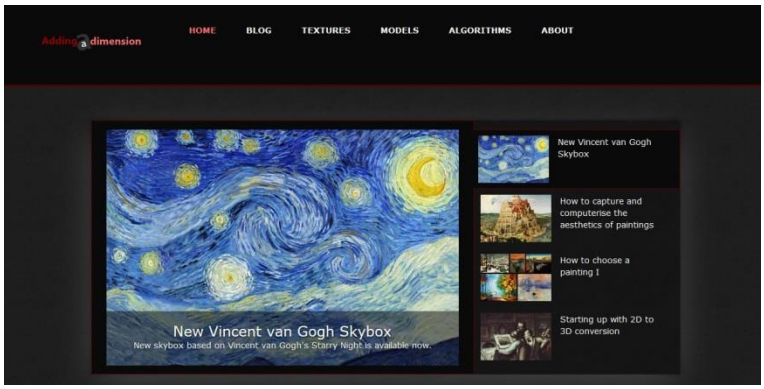


Figure 73: Home page

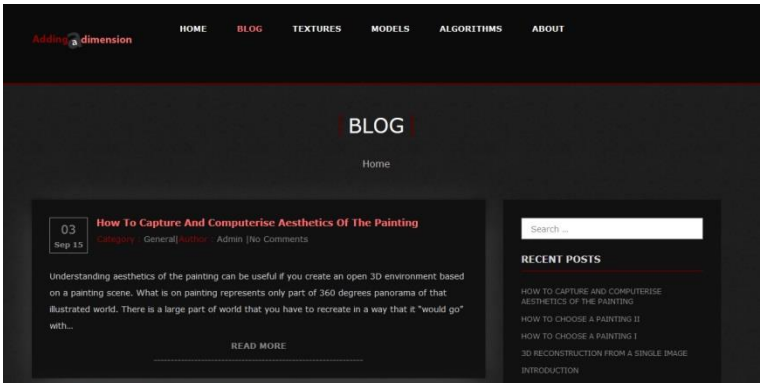


Figure 74: Blog section

The blog section (Figure 74), where information about the conversion process is posted has the same design as the rest of the site, even though it is based on different CSS files created for the WordPress blog templates.

The pages present all products that were created in the course of the project. These products are downloadable and ready for future reuse. Figure 75 illustrates the website section for the 3D models featuring subtle motion graphics when the user hovers over the model.

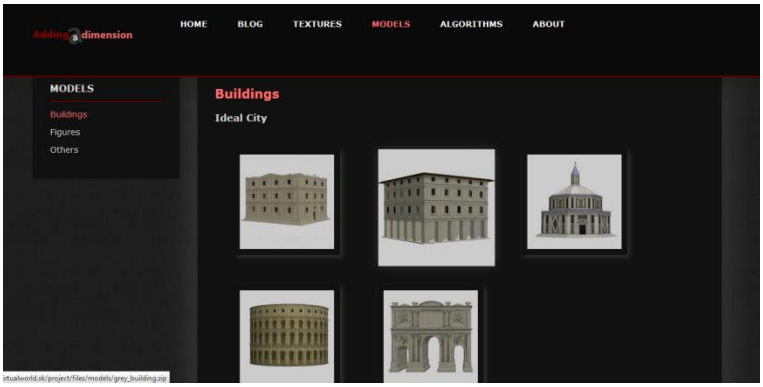


Figure 75: Presenting 3D models

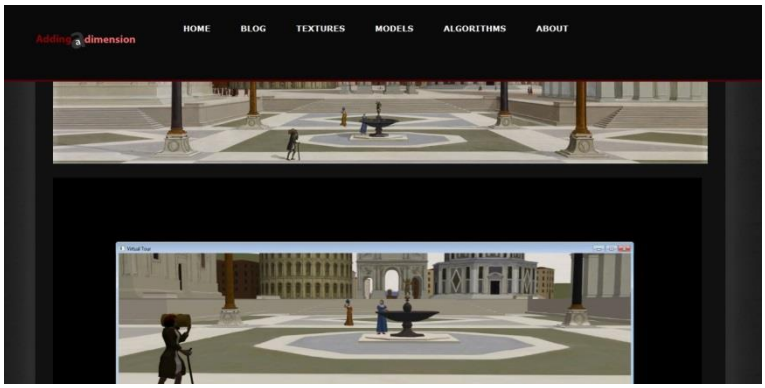


Figure 76: Painting and video on questionnaire page

The questionnaire was created under a separate testing part that isn't accessible from the main menu but rather a link is provided to potential testers so they can access this part. The questionnaire page (Figure 76) contains an original painting, an application video and a form with radio buttons and two text slots.

At the time of report submission the website contained all important models from the painting conversion along with five blog posts about the conversion. The application videos are published in About->Outcomes section

5. Discussion

In this chapter we revisit the objectives defined at the beginning of the project to evaluate accomplishments of the research work undertaken in the project. We then place the research outcomes in the wider perspective of the work done in related fields. To assess the validity and generalisability of the results we need to consider conversion attempts done in the course of our research and limiting factors that informed our decision regarding the ultimate selection of the painting for the project. At the end of this chapter we make some recommendations about possible optimisation and automation of the whole process in the future.

5.1. Objectives Fulfilment

The aim of this project was to provide a proof of concept of a production method that enables a skilled human operator, using specialized tools and techniques, to create a computer graphics application that simulates the walkthrough of a 3D scene, visually based on a chosen painting. To accomplish this aim, several project objectives have been defined. In the next section we analyse fulfilment of these objectives.

5.1.1. Existing Conversion Tools and Software

In the initial stage of this project we performed an extensive research of tools that could possibly simplify the conversion process. We looked for both established software tools and experimental research code. In case of the latter we found several promising papers for which real software solutions were not yet available.

In our conversion trial, we have worked mainly with established software tools, both commercial and open source. In section 4.1.2 software tools for each part of the conversion process are listed, along with several different alternatives that vary in interface and availability / cost.

We specified several pros and cons of the tools but we didn't go into great detail, because differences in user interface are seen quite subjectively by different users. We don't claim our list as definitive since some software tools don't have free trial licensing and therefore they haven't been tried. But we covered most of the conversion tasks with freely available software.

5.1.2. Recommendations for Painting Selection/Rejection

In section 4.1.1 we identified criteria for painting selection with their indicators and examples of the easy and hard paintings to convert. It should be noted that this list was made with a focus on the whole painting conversion, as close to the original as possible, and

creation of the full 3D environment based on the painting. If any of these preconditions are dropped, several criteria may be no longer relevant.

The criteria selection and description are based on conversion attempts of several paintings that were chosen at the beginning as possible candidates and dropped at later in the process. Some paintings were eliminated due to modelling time, some because of performance limitations of the machine, some due to difficulty with distance estimation and others because of impossibility to create 3D objects that would be visually close to the 2D original.

If we ignore painting styles that don't create discrete objects and machine performance issues, the biggest difficulties in the painting conversion are caused by modelling of an amorphous terrain and soft surfaces. Even though Blender provides a tool for physical simulation of cloth behaviour, it creates a cloth, not the particular shape of the cloth that is required. Since the simulation of the cloth behaviour is done in sequence steps, it may be possible to design an algorithm that would catch the correct position of the main ridges on the cloth surface and the small difference can by resolved via sculpting.

A different situation is when one takes only inspiration from the painting without trying to perform a faithful conversion, which makes the whole process of modelling much simpler. However some of our criteria may still be relevant depending on which aspect of the painting is sampled.

5.1.3. Scene Assets

When we started designing the scene assets, we found a relatively large amount of information about 3D modelling, but almost no information about skybox creation. A requirement for seamless texture covering the cube was highlighted. After our first attempt to model the skybox, we discovered a problem with deformation zones and after some analysis we discovered a way to model it and compensate for it. The final skybox for the Ideal City Painting worked well in hiding skybox cube edges.

As mentioned in the previous section we found the problem of amorphous terrain modelling difficult and we tried to approach it in several ways. We attempted to convert the whole terrain section with the Make3D tool (Saxena *et al.*, 2007) (Saxena *et al.*, 2008) but this tool had only limited capabilities for 3D conversion. Its main algorithm works with establishing small homogenous patches in the image and finding depth relationships between different parts of the image. But the painting texture easily confuses this algorithm. There were several paintings where acceptable resulting terrain mesh models were created but with further inspection they turned out to be logically wrong (e.g. algorithm modelled lake surface inclining).

Since a terrain in our chosen painting was mainly flat with one hill in the distance and the rest covered, modelling of terrain wasn't very difficult and final results matched quite precisely when a semi-transparent screenshot of the application was overlaid onto the original painting. The geometric pattern on the ground was particularly helpful for object positioning, and object size estimation, confirmed also by twelve users considering a layout of the scene as 'very accurate' and fourteen users considering sizes of objects as 'very accurate'.

With modelling objects as 3D meshes it turned out that knowing the right 'tricks' for creation of particular types of objects is more important than artistic skills, or to put it another way– lack of artistic skills can be overcome by knowing the right modelling tricks. Objects like statues and human characters in the scene would be particularly hard to model without rigging the base character and placing it into right position. We experienced a drop in modelling time required for acceptable quality characters from 20 hours to 1.5 hour

To create the right shapes of 3D objects was the most difficult part of the modelling. However, only two reviewers considered the shapes 'rather inaccurate', in contrast with six users considering shapes 'rather accurate' and twelve users considering them 'very accurate'.

5.1.4. Scene Completion

Despite an aesthetics guideline definition, we recognise the scene completion task as the one with the least rigorous methodology established. However in comparison with our starting point with no guidelines at all we consider our method satisfactory. Thirteen users responded that the rest of the scene is well fitted with no visible boundaries and seven users considered it as plausible. That doesn't quite respond the question whether we got the paintings aesthetics right but it is hard to question the users about their perception of aesthetics without leading them in certain way.

5.1.5. Camera Modes and Virtual Tour

We created a track for the camera movement that goes around all important assets of the scene. Using splines for that purpose doesn't provide us completely fine controls where all points of the track are placed. On the other hand defining a virtual tour path along the splines offers the user a smooth undisrupted walkthrough.

With three camera views for movement mode (front, left and right) along with the view mode from the top and free camera movement view we provide user enough means to go around and inspect the scene.

5.1.6. Evaluation of the Scene

We performed an evaluation of the scene via the online questionnaire, which enabled us to reach a wider group of users, than we would do if we had to arrange the personal meetings for the potential users. Since the main purpose of this project wasn't a use study we consider our number of twenty users evaluating the application video satisfactory.

Based on users' answers, the scene conversion can be assessed as successful, with only two rather negative answers in one question (objects shapes) and the rest rather positive or completely positive answers with the prevalence of the completely positive. The questionnaire also provided information as to which paintings users would be curious to see in 3D and confirmed our theories that 3D paintings as an educational environment or gaming environment would be interesting for wider audience.

5.1.7. Contribution to the Body of Knowledge

We intended to provide a contribution to the body of knowledge in the form of the Conversion User Guide. We created an online platform where the guide and the project outputs are published with smaller part of content deployed at the time of project submission. At the beginning of the project we planned to have a larger amount of content published at this point in time, however because of continuous learning process throughout the project to the very end, any earlier published materials would probably have had to be revised at this time.

At the completion of our modelling work, there is an opportunity to reflect on all the conversion activities and think about what to share with the rest of the research community.

5.2. Research in Wider Perspective

We tried to use the Make3D tool (Saxena *et al.*, 2007) (Saxena *et al.*, 2008) in our conversion process on many paintings so we can conclude that for the painting conversion this tool is not particularly suitable. If we abstract from painting specifics, Make3D creates a single mesh object over which a texture is orthographically projected. This often creates very visible stretching in textures. In our approach we do split texture into parts or we project only a single part and texture the rest of the objects with seamless texture tiles. The mesh object created wouldn't be useful for a virtual 3D environment because all objects are connected and textured only from one side. However Make 3D is automatic and for certain pictures it creates an interesting fly-around.

In their research, Aubry tries to solve a similar problem of the exact positioning of the 3D objects on 2D depictions as we have encountered (Aubry *et al.*, 2014). They approach it by finding discriminating elements that would help align 2D depiction with an existing 3D model. His research scope is narrower than ours because he is only working with a single architectural object, whereas we had to consider the whole painting space when establishing the painting's viewport. However we used the discriminating elements when we were doing verification of our positioning.

Kholgade use in their research publicly available 3D objects which they align with 2D objects in the image and create 3D objects as a combination of the two with the final appearance of an object that was previously 2D (Kholgade *et al.*, 2014). We used similar technique for extraction, and we were also working with objects symmetries as they did. However our approach is more robust because we used the external models in a very limited way. Instead, we used very generic 3D models that we shaped and sculpted or created from scratch.

Philosophically the closest approach to ours is one presented by Crimisi (Criminisi *et al.*, 2005). They use linear perspective assessment; they establish object sizes and distances relations. In their research paper they have presented Masaccio's Trinity reconstruction based on their art analysis. However that painting contained a reconstruction of only a small semi-enclosed space. Our reconstruction included creation of the whole open space.

5.3. Results Validity and Generalisability

The results we have achieved in this project cannot be generalised for the all paintings as one single group. As we have seen with painting selection criteria, there are many aspects that have an influence on the successful painting conversion. If we abstract from time and performance difficulties, and minor problems caused by complicated textures or small viewing angle, we are still left with the object discreteness requirement, realistic perspective requirement, and scene structure and surface complexity on the medium or low side of the difficulty scale.

We are confident that our methods would be valid for the most city paintings and also open spaces that contain some geometric objects. We haven't managed to find a precise or at least good enough method for the amorphous terrain modelling, but based on our current knowledge of the computer vision area such a task would be beyond the scope of this project.

5.4. Recommendations

We have found a proof of concept for the conversion process of a certain category of paintings. Once we know all the steps required we can start thinking about simplification, optimisation and automation. The most laborious tasks in the process are texture reconstruction and 3D modelling, so we would start looking for potential optimisations in these areas. We have found that Blender software has a python console, which enables mesh data manipulation, so that would be our first future direction to explore on the path to automation.

6. Evaluation, Reflections and Conclusions

This chapter provides an assessment of the whole project, from the choice of objectives and scope to the deployment of its results, with highlights and difficulties experienced in the individual stages. Important lessons learned have been captured and ideas for future work have been formed based on this research experience.

6.1. Evaluation

The project took as a starting point for its problem statement to the task of creating a single view reconstruction of a painting scene in the same way as a photographic scene reconstruction is done. The only difference was that a painting reconstruction can never have a proper verification based on real world measurements.

But what verification could be found for the scene reconstructed from a painting? Besides the pixel by pixel comparison of the one particular view of the scene (which can work even for completely wrong geometry, see Section 2.1.1 – inverse problem), the only verification can come from human observers of the scene. And therefore an understanding of human perception turned out to be crucial for the project.

With this assumption, the scope of the relevant knowledge domain grew enormously. If the project dealt only with the computer vision area of expertise, it would still be more than enough to examine within the given time. So from the expertise perspective, the choice of objectives was very demanding. On the other hand, when considering project deliverables derived from project objectives they were well scoped and self-contained.

The greatest difficulties in the project arose around the terrain modelling task. When a preliminary research of existing tools was done, expectations were formed that an initial model for terrain could be produced automatically and our work would be mainly focused on refinement. This did not eventuate, and we were obliged to turn our attention to paintings with less unstructured terrain.

The biggest surprise in the project was how well 3D modelling tools were designed and how easy it was even for an amateur to work with them. Creation of complicated geometry can be done nearly effortlessly and different parts of objects are easily reusable. There are many modelling tricks shared by professional online, some of which contributed to the successful completion of this project within given time.

As a great achievement is considered the creation of skyboxes, which have considerable impact on virtual environment look and can be reused easily in various projects. Finding the way how to locate a good cutting line, proliferation of the textures and refinement for the final assembly stage constitute an original contribution to the pool of knowledge. The second important achievement is the incorporation of human perception theories into the overall conversion process.

If the project proposal could have been redefined in the light of current understanding its scope would be narrowed down and chosen research problem would be more specific.

6.2. Reflections

From the very beginning, the project was considered to be ambitious by me and my supervisor. How ambitious it was, I began to realise during the first stage when the major part of studies of the relevant domains had been done and the difficulty with single view geometry reconstruction had been discovered. Before the beginning of project, I had only limited knowledge of projection geometry from high school maths and partial knowledge of the research done in human vision and perception.

The major knowledge gathering was accomplished in computer vision field via online courses and literature study. A top-down approach was taken in order to find out what techniques are available and then more focused study was conducted in the areas relevant for the project. Other gains from project are practical skills in 3D modelling. Despite having experience with image editing in Photoshop, I have never used Blender before this project. Difficulties with the numerous unfamiliar tools in Blender were overcome via following online step-by-step learning courses.

The project also provided insight into the research work organisation, which is different from the organisation in typical commercial IT projects. However, the principles of incremental planning and quality management can be successfully applied even in research projects.

Sometimes in research projects it is necessary to perform intentional activities to induce creativity and generation of the new ideas. In this project a quite opposite was true. After the literature survey, a plethora of diverse ideas was available and the real endeavour was to reject unsuitable ones. That had contributed to sharpening of my critical scientific judgement.

There are two major lessons learned from this project. The first one says: you should always know about all important challenges in the field you are going to research. And second one says: if you don't manage to do the first one, you should be really passionate about your research problem, because you are going to spend lot of time studying it.

6.3. Future Work

The allure of the single view geometry reconstruction problem is that it still remains open and therefore lot of improvements can be achieved in the future. With the current know-how, several directions of potential research can be developed.

With more time and resources, a user study can be done to verify perception theories that accompanied our reconstruction process. A virtual environment, such as the one we have created in OpenGL would be ideal for this kind of study because models can be produced easily with slight variations of different parameters and then judged by group of research participants.

A more focused research project can be formed around the problem of unstructured terrain reconstruction and modelling. However, the recommendation from this project would be to choose the source images with at least several classifiable objects in it.

It would be also interesting to create general guidelines for polygon budget assessment when modelling is done from 2D images. That can build upon our recommendations for painting selection, where this factor was partially considered.

With the Conversion User Guide blog and web platform created for this project, there is a potential to make it known and populate it with more objects and learning articles. A current plan is to continue modelling objects from well-known paintings and to attempt more automatization in the whole process. A research work can be done in curve parametrisation of the 2D objects outlines and 3D models adaptation via parametric equations.

6.4. Conclusions

This project intended to provide proof of concept of creating plausible 3D scene from 2D painting. The resulting virtual environment models an open space with complete 3D objects. In comparison with previous research and conversion attempts, this research project provides as an outcome a functional environment based on open source technology that can be used for further purposes, such as for games or as education material.

To overcome the difficulty of the core problem of this project, a wide spectrum of knowledge domains was examined and some unconventional approaches were implemented. The most fascinating theories discovered in the process involved human vision and perception. Dissecting the process of seeing things by human beings uncovers complex and unexpected mechanisms that enable us to deal with inverse problem of vision in everyday life.

In future, an endeavour to reconstruct 3D scene in computer vision and robotics might find useful to rely on knowledge in human perception. It would be interesting to analyse different visual processes and try to simulate them with algorithms. Given the time and evolutionary pressure, human ability to see and interpret has become extremely fine-tuned and can provide lot of inspiration. Of course this wouldn’t have a general applicability in places that aren’t common human habitat such as other planets but unless we take up the path to Type II civilisation the most of machinery will be still down to on Earth.

Glossary

Anisotropic filtering – a technique to enhance quality of textures that are in oblique viewing angles in respect to camera

Back-face culling – a process of determining whether object's polygons are front facing (with anti-clockwise polygon points order) or back facing (clockwise order), and rendering only front facing polygons

High-poly mesh model – a mesh in computer graphics with high number of polygons (in hundred thousands)

Inverse problem – a problem of calculating from a set of observations the causal factors that produced them

Low-poly mesh model – a mesh in computer graphics with relatively small number of polygons (in thousands)

Mesh model – a collection of vertices, edges and faces that defines the shape of 3D object in computer graphics

Primitives – a geometric form of objects in the scene, represented as collections of basic shapes (points, triangles, lines)

Rendering – a process of synthetizing an image from a 2D or 3D model by means of computer program

Scene – a collection of objects in space and their organisation

Shader – a program that is responsible for producing appropriate level of colours within synthesised image

Skybox – a collection of six textured square images that creates seamless surface of the cube representing the outer world in virtual environments

Stereopsis – a perception of depth and 3-dimensional structure obtained on the basis of visual information deriving from using two eyes together

Surface normal – a line or vector that is perpendicular to a given surface

Texture baking – process of transferring lighting, material o geometry information in to an image texture

Vertex – a point that describes the corners or intersections of geometric shape

References

- Adobe Kuler (2015) *Color wheel*. Available at: https://color.adobe.com/create/color-wheel/ (Accessed: 25. June 2015)
- Andrews, S. (2013) 'Closer to reality: photorealism in computer graphics', *Alphr*, 6 December [Online]. Available at: http://www.alphr.com/features/385879/closer-to-reality-photorealism-in-computer-graphics (Accessed: 15 August 2015)
- Aubry, M., Russell, B.C., Sivic, J. (2014**) '**Painting-to-3D model alignment via discriminative visual elements'**,** *ACM Transactions on Graphics (TOG),* 33(2), art. 14 [Online]. Available at: http://dl.acm.org/citation.cfm?id=2591009 (Accessed: 14. April 2015)
- Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B. (2009) 'Patchmatch: a randomized correspondence algorithm for structure image editing', *Proceedings of ACM SIGGRAPH 2009*, 28(3), art. 24 [Online]. Available at: http://dl.acm.org/citation.cfm?id=1531330 (Accessed: 21. April 2015)
- BBC News (2015) *Turning Van Gogh's The Night Cafe into virtual reality*. Available at: http://www.bbc.co.uk/news/technology-32751392 (Accessed: 20. August 2015)
- BetterExplained (2015) *Math lessons that actually explain concepts*. Available at: http://betterexplained.com/ (Accessed: 22. August 2015)
- Blender Foundation (2015) *Blender Reference Manual.* Available at: https://www.blender.org/manual/ (Accessed: 7. September 2015)
- Bootstrap (2015) Available at: http://getbootstrap.com/ (Accessed: 10 July 2015)
- Boudon, G. (2013) 'Understanding a 3D Production Pipeline – Learning the Basics', *Digital-tutors.* [Online] Available at: http://blog.digitaltutors.com/understanding-a-3d-production-pipeline-learning-the-basics/ (Accessed: 28 August 2015)
- Brannan, D.A., Esplen, M.F., Gray, J.J. (2012) *Geometry*. 2nd edn. Cambridge: Cambridge University Press.
- CG Cookie (2015) *Courses – Blender.* Available at: https://cgcookie.com/courses/?fwp_division=blender (Accessed: 20. August 2015)
- Chen, T., Zhu, Z., Shamir, A., Hu, S.M., Cohen-Or, D. (2013) '3-Sweep: extracting editable objects from a single photo', *ACM Transactions on Graphics (TOG),* 32(6), art. 195 [Online]. Available at: http://dl.acm.org/citation.cfm?id=2508378&CFID=711576597&CFTOKEN=50177425 (Accessed: 9. September 2015)
- Cohen, B.J., Wood, D.L. (2000) *The Human Body in Health and Disease.* 9th edn. Philadelphia: Lippincott Williams & Wilkins.
- Collins, R. (2007) 'Lecture Notes', *CSE/EE486 Computer Vision I.*
- Criminisi, A., Kemp, M., Zisserman, A. (2005) 'Bringing pictorial space to life: computer techniques for the analysis of paintings', *Technical report of Microsoft Research.* [Online]. Available at: http://msr-waypoint.com/pubs/67264/criminisi_chart_yb2005.pdf (Accessed: 10. August 2015)
- Dawson, C.W. (2009) *Projects in Computing and Information Systems.* 2nd edn. Harlow: Pearson Publication Ltd.
- Debevec, P.E., Taylor, C.J., Malik, J. (1996) 'Modeling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach', *Proceedings of ACM SIGGRAPH 1996*, pp. 11-20 [Online]. Available at: http://dl.acm.org/citation.cfm?id=237191 (Accessed: 20. April 2015)

- Gimpel3D (2015) Available at: http://www.gimpel3d.com/ (Accessed: 18. April 2015)
- Great Britain. Parliament. House of Commons. (2000) *Copyright Law* [Online]. Available at: https://www.copyrightservice.co.uk/copyright/p01_uk_copyright_law (Accessed: 18. April 2015).
- Hartley, R., Zisserman, A. (2004) *Multiple View Geometry in Computer Vision.* 2nd edn. Cambridge: Cambridge University Press.
- Hubel, D. H., Wiesel, T. N. (2005) *Brain and Visual Perception: The Story of a 25-Year Collaboration.* New York: Oxford University Press.
- 'Image segmentation' (2015) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Image_segmentation (Accessed: 30. August 2015)
- 'Information architecture' (2015) *Wikipedia.* Available at: https://en.wikipedia.org/wiki/Information_architecture (Accessed: 26. August 2015)
- Itti, L., Koch, C. (2000) 'A saliency-based search mechanism for overt and covert shifts of visual attention' *Vision Research,* 40(10), pp. 1489-1506 [Online]. Available at: http://www.sciencedirect.com/science/article/pii/S0042698999001637 (Accessed: 30 August 2015)
- Kemp, M. (1990) *The Science of Art: Optical themes in western art from Brunelleschi to Seurat.* London: Yale University Press.
- Kholgade, N., Simon, T., Efros, A., Sheikh, Y. (2014**) '**3D object manipulation in a single photograph using stock 3D models', *ACM Transactions on Graphics (TOG)*, 33(4) art. 127 [Online]. Available at: http://dl.acm.org/citation.cfm?id=2601097.2601209 (Accessed: 12. April 2015)
- Krantz, J. H. (2010) *Vision and art*. Available at: http://psych.hanover.edu/Krantz/art/ (Accessed: 14. August 2015)
- Krug, S. (2006) *Don't Make Me Think! A Common Sense Approach to Web Usability.* 2nd edn. Berkley: New Riders Publishing.
- Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A., (2003**) '**Graphcut textures: image and video synthesis using graph cuts**'**, *Proceedings of ACM SIGGRAPH 2003*, 22(3), pp. 277-286 [Online]. Available at: http://dl.acm.org/citation.cfm?id=882264 (Accessed: 22. April 2015)
- Lai, Y.K., Hu, S.M., Martin, R.R. (2009**) '**Automatic and topology-preserving gradient mesh generation for image vectorization**',** *Proceedings of ACM SIGGRAPH 2009*, 28 (3), art. 85 [Online] Available at: http://dl.acm.org/citation.cfm?id=1531391 (Accessed: 19. April 2015)
- Land, E. (1959) 'Experiments in color vision', *Scientific American*, 200(5), pp. 84-89 [Online] Available at: http://www.millenuvole.org/f/Fotografia/Per-quali-ragioni-vediamo-i-colori/edwin_land_1959.pdf (Accessed: 23. August 2015)
- LearnCpp.com (2015) *Tutorials to help you master C++ and object-oriented programming.* Available at: http://www.learncpp.com/ (Accessed: 22. August 2015)
- Ma, Y., Soatto, S., Kosecká, J., Sastry, S. (2005) *An Invitation to 3D Vision: From Images to Geometric Models.* 2nd end. New York: Springer.
- Malik, J. (2012) *Computer Vision.* Available at: https://www.youtube.com/playlist?list=PLc0IeyeoGt2xtmfaF2ST_uNdeptre3f9s (Accessed: 23. July 2015)
- Masters, M. (2014) 'What's the Difference? A Comparison of Modeling for Games and Modeling for Movies', *Digital-Tutors*, 2014 [Online]. Available at: http://blog.digitaltutors.com/whats-the-difference-a-comparison-of-modeling-for-games-and-modeling-for-movies/ (Accessed: 19. April 2015)

- Matlab (2015) *Image Processing Toolbox.* Available at: http://uk.mathworks.com/help/images/index.html (Accessed: 3. September 2015)
- Niebur, E. (2007) 'Saliency map', *Scholarpedia* [Online]. Available at: http://www.scholarpedia.org/article/Saliency_map (Accessed: 25. August 2015)
- 'Noise reduction' (2015) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Noise_reduction#In_images (Accessed: 30. August 2015)
- Oates, B.J. (2006) *Researching Information Systems and Computing*. London: SAGE Publications Ltd.
- Office of Government Commerce. (2009) *Managing Successful Projects with PRINCE2™*. 5th edn. London: The Stationery Office.
- Pellacini, F. (2009) *Projects in Digital Arts. Available at: http://pellacini.di.uniroma1.it/courses/projects09/projects09.html* (Accessed: 28. August 2015)
- Polack, T., LaMothe, A. (2003) *Focus On 3D Terrain Programming.* Cincinnati: Premier Press.
- Politzer, T. (2008) 'Vision is our Dominant Sense', *Brainline.org*, November [Online]. Available at: http://www.brainline.org/content/2008/11/vision-our-dominant-sense_pageall.html (Accessed: 13 August 2015)
- Purves, D. (2010) *Brains: How They Seem to Work*. New Jersey: Pearson Education Inc.
- Purves, D. (2015) *Visual Perception and the Brain.* Available at: https://www.coursera.org/course/visualpercepbrain (Accessed: 22. August 2015)
- Radke, R. (2014) *Computer Vision for Visual Effects. Available at:* https://www.youtube.com/playlist?list=PLuh62Q4Sv7BUJlKlt84HFqSWfW36MDd5a (Accessed: 24, August 2015)
- Radke, R. (2015) *Intro to Digital Image Processing.* Available at: https://www.youtube.com/playlist?list=PLuh62Q4Sv7BUf60vkjePfcOQc8sHxmnDX (Accessed: 18. August 2015)
- Ramachandran, V. S. (2005) *Phantoms in the Brain: Human Nature and the Architecture of the Mind.* 3rd edn. London: Harper Perennial.
- 'Rapid application development' (2015) *Wikipedia.* Available at: https://en.wikipedia.org/wiki/Rapid_application_development (Accessed: 17. August 2015)
- Sacks, O. (2010) *The Mind's Eye.* Croydon: CPI Group (UK).
- *Saliency Toolbox* (2015) Available at: http://www.saliencytoolbox.net/ (Accessed: 4. September 2015)
- Saxena, A., Schulte, Chung, S.H., Ng, A.J. (2007) '3-D Depth Reconstruction from a Single Still Image', *International Journal of Computer Vision (IJCV)*, 76 (1), pp. 53-69 [Online]. Available at: http://www.cs.cornell.edu/~asaxena/learningdepth/saxena_ijcv07_learningdepth.pdf (Accessed: 10. April 2015)
- Saxena, A., Sun, M., Ng, A.J. (2008) 'Make3D: Learning 3-D Scene Structure from a Single Still Image', *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI),* 31 (5), pp. 824-840 [Online]. Available at: http://www.cs.cornell.edu/~asaxena/reconstruction3d/saxena_make3d_learning3dstructu re.pdf (Accessed: 10. April 2015)
- Sedgwick, H. A. (2004). 'Space perception', in Goldstein, E. B. (ed.) *Handbook of Perception and Human Performance*. New York: John Wiley & Sons, pp. 128-167.

- ‘Sense’ (2015) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Sense (Accessed: 9. September 2015)
- Shreiner, D., Sellers, G., Kessnich, J.M., Licea-Kane, B.M. (2013) *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 4.3*. 8th edn. Addison-Wesley.
- Shroff, A. (2011) *An Eye on Numbers: A Ready Reckoner in Ophthalmology*. PostScript Media Pvt. Ltd.
- Shuja, A.K., Krebs, J. (2008) *IBM Rational Unified Process Reference and Certification Guide.* IBM Press.
- Slabaugh, G. (2015) ‘Lecture slides’, *INM376 Computer Graphics.*
- Solso, R. L. (1996) *Cognition and the Visual Arts.* Massachusetts: The MIT Press.
- Sundstedt, V., Whitton, M., Bloj, M. (2009) **‘**The whys, how tos, and pitfalls of user studies’, *Proceedings of ACM SIGGRAPH 2009,* ACM Digital Library [Online]. Available at: http://dl.acm.org/citation.cfm?id=1667264 (Accessed: 19. April 2015)
- Szeliski, R. (2010*) Computer Vision: Algorithms and Applications*. [Online]. Available at: http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf (Accessed: 26. August 2015)
- ‘Website wireframe’ (2015) *Wikipedia.* Available at: https://en.wikipedia.org/wiki/Website_wireframe (Accessed: 26. August 2015)
- Wildberger, N. J. (2009) *Wild Trig.* Available at: https://www.youtube.com/playlist?list=PL85A84C3580CADD64 (Accessed: 16 July 2015)
- Wolff, D. (2011) *OpenGL 4.0 Shading Language Cookbook.* Birmingham: Packt Publishing Ltd.
- Zucker, S., Harris, B. (2015) *How one-point linear perspective works.* Available at: https://www.khanacademy.org/humanities/renaissance-reformation/early-renaissance1/beginners-renaissance-florence/v/how-one-point-linear-perspective-works (Accessed: 26. August 2015)

Appendix A – Project Proposal

Introduction

3D modelling is widely used across different industries including movies, animation and gaming. Techniques applied in movie and game modelling are similar, however there is a big difference in procedures and model quality limitations given by real-time rendering required in computer games and simulations (Masters, 2014).

The last decade of development in computer graphics however brought significant advances for an ordinary graphics developer. OpenGL became one of the most popular APIs for 3D graphics thanks to, among the other reasons, shader programs giving more flexibility and power to the hands of developer (Wolff, 2011, p. 7). Together with ever-increasing computational performance it enables creation of high quality graphics environments for games and simulators.

The other direction of progress in 3D graphics involves scientists trying to add the third dimension to different elements of a 2D image source. Algorithms adding depth to the image were invented with partially promising results (Saxena *et al.*, 2007) and also methods for modelling various single artefacts.

Research Question

Based on gaining experience with OpenGL in the course Computer Graphics module and having a lifelong art appreciation, the following research question was formulated:

Is it possible to perform conversion of a painting into fully contained 3D graphics scene as one person's job, using commonly available software and tools without losing the artistic experience?

With all publicly available research and tools, with high performance hardware in average computers maybe it is time to add one more dimension and take artistic experience to new level.

Aim

The aim of this project is to provide the proof of concept of a production method that enables a skilled human operator, using specialized tools and techniques, to create a computer graphics application, which will simulate the walkthrough of a 3D scene, visually based on a chosen painting.

Objectives

Objective	Testable Result
Identify and evaluate existing tools and software[13] that can be used in conversion process	List of tools and software with the pros and cons of their usage
Identify criteria for painting selection	Specified recommendations for painting selection
Design and develop visually pleasing and accurate individual components of the scene	Skybox, terrain, meshes based on 2D representation from painting
Create plausible complementary parts of the scene to those on painting	Complete seamless 3D scene
Design interesting track around the scene and versatile camera modes to present the scene	Different modes of camera view regulated by users with possibility of virtual tour
Perform evaluation of final scene by wider group	Evaluation of the scene via questionnaire
Contribute to "world's body of knowledge" (Dawson, 2009, p. 17)	Learning material documenting the production method that would allow future users to try it

Table 1. Project Objectives

Work Products

At the end of the project, it is intended to deliver following products[14]:

- Computer graphics application – designed for Windows platform using OpenGL API together with OpenGL Shading Language and other related C++ libraries. Application will provide virtual tour around the painting environment, enabling users to interact with it – changing the speed of tour, setting different camera views and enabling narrative mode with information about the painting. Painting environment should be true to original and there should be no obvious boundary between parts shown on painting and rest of the scene.
- Conversion User Guide – that will document conversion process in order to provide information for future users who would like to repeat the process. It will be designed as modern learning material with links to used tools and resources, and it will also contain information about alternative tools and techniques that were examined during the research process but weren't used at the end.

[13] One of the prerequisites is license allowing free use or at least use for research

[14] According to our project management methodology defined in Project Management section, scope of the project is determined by *Project Product Description*. *Project Product* is described in Plan of Work section.

Project Beneficiaries

This project expects to produce useful outputs for following groups of beneficiaries:

- Game developers – the Conversion User Guide can help game developers who lack inspiration in graphics design and want to focus on game functionality to create scene for their game using 2D image sources. Individual elements of the application design (skybox, textures, and meshes) will also be made publicly available so they can be reused in other computer graphics projects.
- Computer graphics students – the Conversion User Guide will serve students who want to explore modern computer graphics features beyond their basic computer graphics course. Especially City University London's students can benefit from the additional information from the Guide since it will build upon topics taught in the computer graphics module. Source code of the application can also be provided.
- Art educational institutions and galleries – application can be offered to these organizations for education or entertainment. It will be easily possible to add narrative about the painting which can guide virtual tour around it.
- Researcher – as well as from outputs of project that can be included into work portfolio, researcher will benefit from the whole research experience by gaining insight into academic research procedures, learning how to create 3D meshes, improving OpenGL knowledge and keeping C++ programming skills fresh.

Critical Context

The scope of this project spans several areas of computer graphics. A basic requirement for creation of the scene is to know about possibilities of the modern OpenGL programming interface and OpenGL Shading Language (GLSL). In order to design procedures for development of separate components of the scene we need to explore 2D to 3D conversion techniques and tools. Special attention needs to be paid to mesh creation and a separate topic is reconstruction of hidden and non-existing parts of the painting necessary for complete scene assembly.

As we are attempting a novel approach in reproducing a 3D scene from a 2D source, we wanted to have an overview of established procedures and algorithms in the area as well as an account of recent progress and tools. Thus, although some of the following papers may be slightly dated, they remain highly relevant; conversely some of the newest research hasn't permeated the wider community.

OpenGL and GLSL

OpenGL is a hardware-independent application programming interface consisting of hundreds[15] of distinct commands that you use to define the objects and operations needed to produce interactive 3D applications (Shreiner *et al.*, 2013). OpenGL 4.x together with GLSL and

[15] Depends on 4.x version of OpenGL.

related libraries provide a wide range of functionalities useful for reproducing painting scenes, such as transformations, texture and multi-texture mapping, anisotropic filtering, height maps, different lighting models, shadow mapping, reflection, blur, alpha maps, fog and import of meshes of various formats (Slabaugh, 2015).

2D to 3D Conversion and Mesh Creation

One of the crucial problems of converting a 2D image to a 3D scene is to determine appropriate positions of the individual elements in the scene when adding depth. Make 3D is a tool that resulted from the research of depth reconstruction from a single still image (Saxena *et al.*, 2007) followed by 3D scene structure estimation (Saxena *et al.*, 2008). A focus of the research was on unstructured environments (no special assumptions are made about the structure of the scene). The method of conversion consisted of image segmentation into small, approximately planar surfaces (similar to graphics representation of objects in OpenGL) and finding out 3D position and orientation using a machine learning algorithm. Thereafter, a 3D mesh model of the scene could be reconstructed. Based on results of testing and videos of 'fly-arounds' of the created 3D model, the tool can be potentially useful in our research for terrain modelling.

Most of the research concerning 2D to 3D transformation has narrower scope and is focused on specific technique of conversion. A novel approach was introduced with the alignment of 2D depictions, including paintings of architectural motifs (e.g. city square with monument), with the 3D model of the site (Aubry *et al.*, 2014). It solves problem of painting alignment, with many elements on the scene and created in variations of artistic styles, via small sets of visually discriminative elements. The disadvantage of this method is the need for 3D models of the buildings from the painting, so it would be useful for our research if our painting contained only known places for which 3D models of its landmarks already exist. When having more than one image source, effective methods have been developed for architectural scene modelling (Debevec *et al.*, 1996).

Using much simpler existing 3D models, some research has been done in creating object models from photographs by aligning and joining visible parts of the object from in an image with similar 3D models to recreate the non-visible parts (Kholgade *et al.*, 2014). Possible mismatches between the image and the model are corrected using pixel information from the image while keeping approximate object symmetries and also considering illuminations. The practicality of using this approach in our research depends on the availability of relevant 3D models.

In general, techniques for mesh creation vary depending on complexity of surfaces, textures and object realism. Promising research has been done in many areas concerning object 3D modelling, including an automatic gradient mesh generation for whole image (Lai *et al*, 2009), significantly speeding the process and removing need for manual interaction. In parallel, advanced tools for 3D modelling have been developed, e.g. Gimpel3D that is both versatile and free for use (Gimpel3D, 2015).

Image Reconstruction

Since the source of information about a 3D scene is single image, by isolation of the foreground objects into 3D models, missing parts of the objects in the back of the scene will become visible when the viewing angle changes. Therefore it is necessary to produce replacements for missing parts of various items in the image. For this purpose, many algorithms to find nearest neighbour matches of missing parts have been developed, some of them obtaining efficiency through random sampling to find good matches and their subsequent propagation (Barnes et al., 2009). In the area of efficient seamless recreation of larger images from smaller samples, successful research has been conducted for more than 10 years (Kwatra *et al.*, 2003).

Methods & Tools for Analysis & Evaluation

A research strategy for this project is the design and creation, and the contribution made to the knowledge will be the method (documented in the guide) and instantiation (application) (Oates, 2006, p. 108). For application development it is necessary to employ a structured and well defined software development methodology. Because of relatively high uncertainty and technological nature of this project, iterative and incremental method seems the most appropriate. Representing this type of method, RUP (**R**ational **U**nified **P**rocess) provides a customizable framework for a specific project, defining phases of the project and process disciplines (Shuja and Krebs, 2008). The whole development, from requirements analysis to evaluation, will be influenced by the artistic nature of this project (Oates, 2009, p. 118-119).

Based on RUP phase definitions, the project's Inception Phase tasks will be finished with this project proposal approval[16], so the project course will start with the Elaboration Phase. Part of the analysis in this phase will include defining criteria for painting selection, testing the tools for mesh creation and search for tools that can simplify modelling of individual elements of the scene. A result of this phase should be a clear vision of procedures and techniques that will lead to the creation of individual components of the scene in application. Evolutionary prototypes as a byproduct of practicing work with tools may be developed as well. Documents produced in this phase will also include moodboard with colours and layout.

The Construction Phase will cover the whole application development process and most of the user guide creation. Elements of the scene will be developed starting with environmental ones first (skybox, terrain), followed by individual meshes, lighting and special effects if necessary. In graphics project like this, users' involvement in evaluation is important (Sundstedt et al., 2009). Because of the timeframe of this project and its primary focus on design and creation, and no budget, a compromise solution will be small focus group of five participants, who are already known to researcher, receiving instructions and performing tasks of evaluation of separate components over the internet. Bias of participants' selection can be overcome by not using ordinal scale for single option evaluation, but rather giving participants more options to choose, as well as collecting qualitative data. An optimal balance of visual quality and performance will be determined by results of application performance testing.

[16] Requirements will be defined before the project starts as a part of completion of the Inception Phase.

The Transition Phase will include an evaluation of the complete scene by a bigger group of participants via a questionnaire. A call for participation will be published through various channels. Respondents will be contacted via email, watch a video and respond to the questionnaire online. However, because of the artistic nature of final product, there will be only few objective criteria to measure (Oates, 2009, p. 119) and qualitative data will also be collected.

The evaluation of the project can be performed at several levels. Basic success would correspond to the production method reliably producing scenes of reasonable visual accuracy. At the next level, success would correspond to a positive reaction gained from the questionnaire. The ultimate measure of success, which won't be possible within timeframe of the project, would be real use of the user guide by graphic developers or use of the application by organisations working with art, such as galleries and educational institutions.

Tools and Production Environment

The application will be written on Windows platform, using Microsoft Visual Studio 2013 IDE. As computer graphics API OpenGL 4.0 was chosen because of its range of features, flexibility and past experience of researcher with it. OpenGL shading language will be part graphics development toolbox as well. The main programming language will be C++ for graphics part as well as for potential algorithms for image processing. C++ is relatively low level, has wide range of libraries for working with images and again it is well known to the researcher. For mesh creation, currently chosen tool is Blender but there is a possibility of change of this decision in the early stage of project. For texture sampling and modification Adobe Photoshop will be used by researcher but in the guide for users GIMP alternative will be used as well[17]. As a version control system for source code will be used Atlassian Bitbucket.

Project Management

Project management of this project will be based on PRINCE2™ (**Pr**ojects **in** a **C**ontrolled **E**nvironment)[18] project management method (Managing Successful Project with PRINCE2™, 2009). The reason for this is that PRINCE2 is a generic method that can be applied to any project (even research one), it provides tailoring guideline for small projects (Managing Successful Project with PRINCE2™, 2009, p. 221), it fully covers important areas of project management according to project proposal requirements (planning, risk management, project products, benefits), and it complements well iterative and incremental development (it is possible to map phases/iterations to the project stages and plan & control accordingly).

Based on PRINCE2 methodology, this project proposal will be considered as *Project Initiation Documentation*, which is used to gain authorisation for the project to proceed. In the final project report, it can be used as base for assessment of actual project performance against the original estimation (Managing Successful Project with PRINCE2™, 2009, p. 162). We will consider Introduction section of this document as *Business Case* justification and the other

[17] We won't consider Adobe Photoshop as commonly available software.
[18] All PRINCE2 related terminology is marked with italics in the text of this document.

required themes are cover within this section and Plan of Work section. *Project Manager* will be the person doing the work and project supervisor will act as *Executive* representing the *Project Board* (Managing Successful Project with PRINCE2™, 2009, p. 222-223).

Communication Management Strategy

Internal communication will include communication with the project supervisor, both in form of meetings and emails. Status of project products will be available to project supervisor via www.virtualworld.sk website and will be reported continually. At the end of each project stage, *End Stage Report* and next *Stage Plan* will be given to the supervisor. In case of stage deviating from plan beyond tolerances agreed with supervisor, an *Exception Plan* will be provided to supervisor.

External communication will include communication with research participants in evaluation part of the project. This communication will be conducted via City University London email and designed website where questionnaire will be embedded.

Configuration Management Strategy

Configuration management is necessary to keep a track of the product development and changes in products in the course of the project. For that reason *Configuration Item Record* will be created to keep the track of product status, version and variants. For source code tracking, a version control system will be used.

Risk Management Strategy

Risk management will include systematic identification and assessment of the risks[19] and implementation he response procedures to prevent or mitigate them (Managing Successful Project with PRINCE2™, 2009, p. 77). All risks with their ID, description, probability, impact, expected value and response procedures will be recorded in a *Risk Register*. *Risk Register* will be updated continually with addition of new risks, reassessment of probability and value of existing risks, modification of response procedures, and removal of the risks that are no longer active.

Since this project doesn't have budget and financial costs cannot be implicated in analysis, we will assess impact of the realization of risk in terms of required additional working time – man-days. A scale for both probability and impact will be set within the range 0 – 1. Probability scale will be derived from percentages and impact will reflect man-days value proportionate to total estimated time needed for project completion[20]. Values of the risk are determined by multiplying probability and impact. Based on value, we will rank risks in non-linear way as low, medium and high.

[19] For the purpose of this project, we will consider only threats and we will omit opportunities.
[20] Estimated time given in Individual Project Reference Card was 600 hours.

Low **value < 0.04** Medium **0.04 ≤ value ≤ 0.1** High **value > 0.1**

ID	Risk Description	Prob.	Imp.	Value	Risk Response
1	Falling behind first stage due to prolonged exam preparation	0.35	0.1	**0.035**	Define more strict limits for studying hours; Try more efficient studying methods to spare time
2	Overambitious project objectives within given timeframe	0.3	0.4	**0.12**	Chose less crowded painting with fewer non-geometric shapes; Look for quick wins for each components
3	Spending too much time with exploration of available tools	0.4	0.15	**0.06**	Narrow down options to most promising tools by testing
4	Insufficient understanding of certain OpenGL features	0.1	0.3	**0.03**	Study for exam; Practise on existing template; Consult supervisor
5	Slow learning curve of mesh creation software	0.35	0.6	**0.21**	Consult supervisor; Use online tutorials; Find meshes that can be partially reused
6	Created scene program will have performance issue due to complicated meshes	0.5	0.6	**0.3**	Do the continuous testing with adding individual mesh to the scene and adjust design accordingly; Consult supervisor
7	Programming difficulties with algorithm for seamless texture creation	0.15	0.2	**0.03**	Research available online C++ libraries for image processing
8	Difficulties with design of missing bits of the painting for the scene	0.3	0.4	**0.12**	Get inspiration from painting restoration techniques; Explore texture generation tools and techniques
9	Failing to assemble whole scene	0.2	0.3	**0.06**	Keep strictly to project plan; Monitor early signs of emergency
10	Difficulty with finding participants for questionnaire[21]	0.3	0.5	**0.15**	Start early; Try different channels for call for participation
11	Data storage or production hardware failure	0.1	0.2	**0.02**	Back-up regularly on external hard-drive; Check performance of other computer
12	Slow progress with writing the final project report	0.1	0.2	**0.02**	Keep Daily Log as information source; Start early

Table 2. Risk Register

[21] Impact of this risk is not fully based on man-days

Quality Management Strategy

A purpose of quality management is to ensure that products meet defined requirements and intended benefits of the project can be achieved. A scope of quality is defined by *Project Product* breakdown structure. Quality criteria for each product will be defined as part of *Product Description* before starting the work on product. When finished, each product will be evaluated against those criteria.

Plan of Work

The main principle of PRINCE2 planning activity is a product based planning. That means project products are identified first and subsequently activities required to deliver these products, together with dependencies and resources, are determined (Managing Successful Project with PRINCE2™, 2009, p. 64). For that reason, we will first perform *Project Product* breakdown structure.

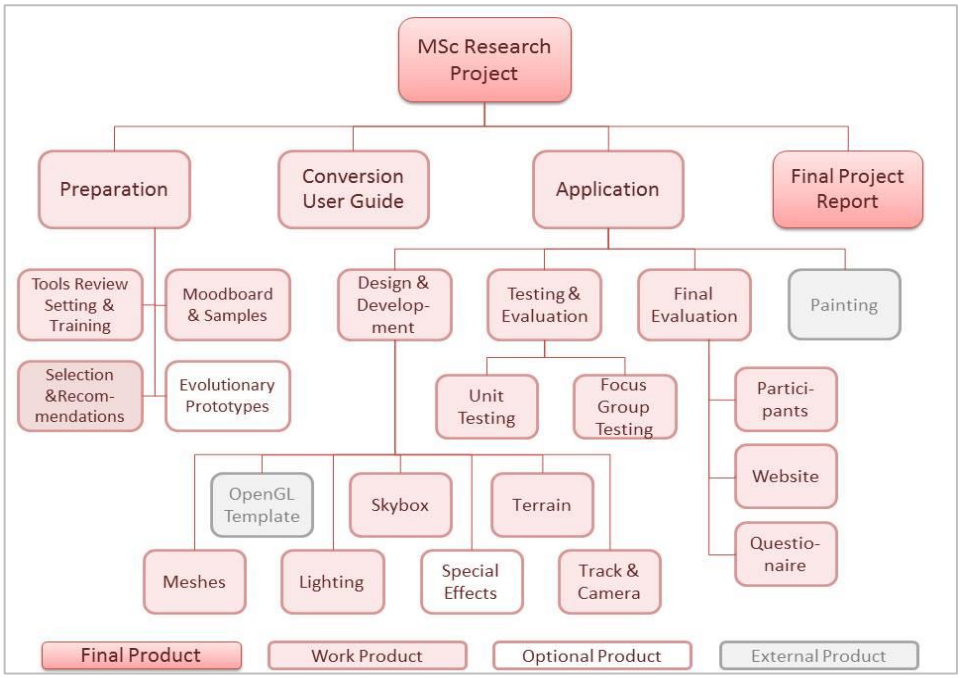
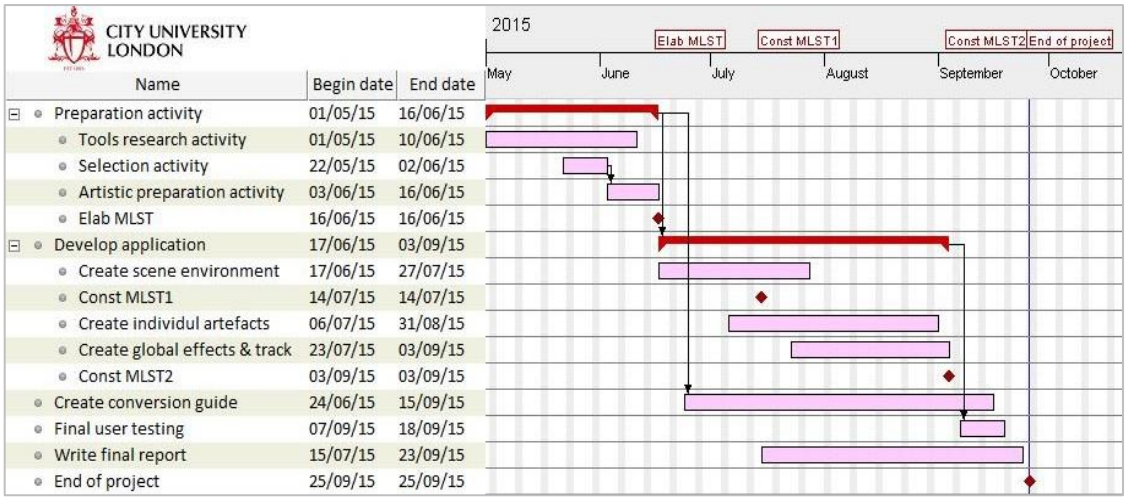


Figure 1. Project Product breakdown structure

After identifying the dependencies between the products, it is possible to proceed with Project Plan. Since it is not possible to plan the entire project in detail at the beginning, Project Plan contains high-level information and should be adjoined with a more detailed Stage Plan for each subsequent stage. It is a common practice to define the main milestones in the project at the end of each phase of development life-cycle[22]. Because of the Construction Phase taking the most of the project's time, for better management it is useful to split it into two iterations with the separate milestones.

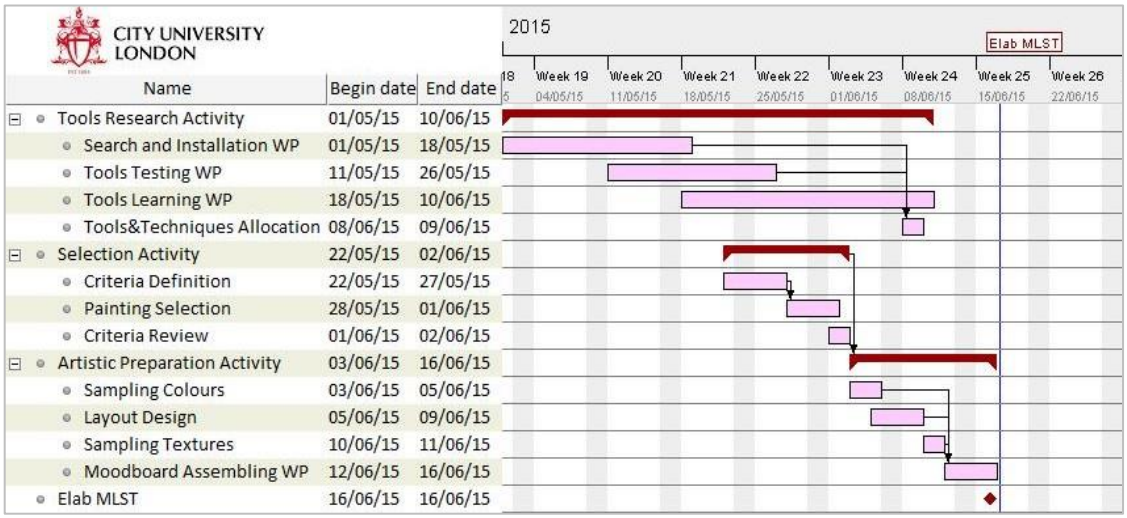
[22] In this project, the Inception Phase ends with approval of project proposal so it is displayed in the plan.



Name	Begin date	End date
Preparation activity	01/05/15	16/06/15
Tools research activity	01/05/15	10/06/15
Selection activity	22/05/15	02/06/15
Artistic preparation activity	03/06/15	16/06/15
Elab MLST	16/06/15	16/06/15
Develop application	17/06/15	03/09/15
Create scene environment	17/06/15	27/07/15
Const MLST1	14/07/15	14/07/15
Create individul artefacts	06/07/15	31/08/15
Create global effects & track	23/07/15	03/09/15
Const MLST2	03/09/15	03/09/15
Create conversion guide	24/06/15	15/09/15
Final user testing	07/09/15	18/09/15
Write final report	15/07/15	23/09/15
End of project	25/09/15	25/09/15

Figure 2. Project Plan

With the outlined *Project Plan* and defined milestones we can establish four stages of the project, each ending with the project milestone. For every *Stage Plan*, high-level activities will be broken down into *Work Packages* and tasks. A plan for each stage will be created before it starts. The first stage will include *Work Packages* and tasks of the tools research activity, selection activity and artistic preparation activity.



Name	Begin date	End date
Tools Research Activity	01/05/15	10/06/15
Search and Installation WP	01/05/15	18/05/15
Tools Testing WP	11/05/15	26/05/15
Tools Learning WP	18/05/15	10/06/15
Tools&Techniques Allocation	08/06/15	09/06/15
Selection Activity	22/05/15	02/06/15
Criteria Definition	22/05/15	27/05/15
Painting Selection	28/05/15	01/06/15
Criteria Review	01/06/15	02/06/15
Artistic Preparation Activity	03/06/15	16/06/15
Sampling Colours	03/06/15	05/06/15
Layout Design	05/06/15	09/06/15
Sampling Textures	10/06/15	11/06/15
Moodboard Assembling WP	12/06/15	16/06/15
Elab MLST	16/06/15	16/06/15

Figure 3. First Stage Plan

In case of time deviation of the actual project course from the current *Stage Plan* beyond tolerance agreed with the supervisor, an *Exception Plan* will be produced to replace the current *Stage Plan* and if it is necessary also the *Project Plan*.

Ethical, Professional & Legal Issues

Ethical Issues

Part of the project research will be evaluation of the computer graphics application product through the questionnaire. Ethical implications of work with participants are covered in Ethics Review Form and Participant Information Sheet. In addition, responsibilities of ethical researcher will be considered – research doesn't require any personal questions causing intrusion to participants or hiding from participants a true purpose of the research (Oates, 2006, p. 60).

Professional Issues

Since this research project intend to build on existing research in the area, credits to the respective authors will be given in the final project report.

Legal Issues

In our research we will choose source painting from a selection whose copyright has already expired. According to UK copyright law an artistic work category which includes paintings is protected by copyright "70 years from the end of the calendar year in which the last remaining author of the work dies. If the author is unknown, copyright will last for 70 years from end of the calendar year in which the work was created, although if it is made available to the public during that time, (by publication, authorised performance, broadcast, exhibition, etc.), then the duration will be 70 years from the end of the year that the work was first made available." (Great Britain. Parliament. House of Commons, 2000).

Appendix B – Project Product Description

Product ID	CGA1
Product Title	Computer Graphics Application
Product Description	Application that provides a virtual tour around the painting 3D environment.
Product Form	Executable file packed with all sources
Requirements	Application provides virtual tour around 3D environment visually based on 2D painting. Application enables user to change speed of the tour and viewing modes. Application runs on Windows OS. Application frame rate on testing machine is higher than 40 frames per second. Polygon budget for the application is between 600,000 – 800,000
Development Skills Required	C++ programming, OpenGL programming, GLSL programming
Technology Required	OpenGL library, GLSL library, C++ compiler, MS Visual Studio IDE, Windows OS
Product Dependencies	CGA2, CGA12, CGA13, CGA14, CGA15, CGA16, CGA17

Product ID	CGA2
Product Title	OpenGL Environment Template
Product Description	Template from INM376 Computer Graphics module that provides basic environment framework.
Product Form	Project folder with basic classes, shader programs, and external libraries.
Requirements	Template files are grouped (header files, source files, shader programs, external libraries) and accessible via MS Visual Studio IDE. Template contains all basic classes (specified in Appendix C). Template is cleaned from unnecessary learning code. Template enables free camera move around the environment.
Development Skills Required	C++ programming, OpenGL programming, GLSL programming
Technology	OpenGL library, GLSL library, C++ compiler, MS Visual Studio IDE,

Required	Windows OS
Product Dependencies	-

Product ID	CGA3
Product Title	Conversion Tools List
Product Description	List of explored tools and libraries with their pros and cons
Product Form	Table
Requirements	Contains only generally accessible software tools with maximum yearly license fee around £200.
Development Skills Required	-
Technology Required	Windows OS, Linux OS
Product Dependencies	-

Product ID	CGA4
Product Title	Production Environment Setting
Product Description	Installed tools and set up libraries that will be needed in conversion process.
Product Form	Installation
Requirements	Tools are working on Windows or Linux platform.
Development Skills Required	System administration
Technology Required	Windows OS, Linux OS
Product Dependencies	CGA3

Product ID	CGA5
Product Title	Painting Selection Criteria
Product Description	List of recommendations for painting selection.
Product Form	Table

Requirements	Concise criteria with difficulty indicator for painting selection.
Development Skills Required	Modelling and computer graphics knowledge
Technology Required	-
Product Dependencies	-

Product ID	CGA6
Product Title	Painting
Product Description	Painting that is source for 3D scene modelling.
Product Form	JPEG image
Requirements	Public domain image 1000px at least in one dimension Feasible for conversion (according to criteria) within given timeframe and computing power Painting contains semi-structured open space
Development Skills Required	-
Technology Required	-
Product Dependencies	CGA5

Product ID	CGA7
Product Title	Processed and Dissected Image
Product Description	Image source after application of image processing algorithms and segmentation process, stored in layers.
Product Form	PSD image
Requirements	All damaged edges removed. Colours of the image sampled. Clean segmantation (removing all pixels that don't belong to object) of components that will need full texture recovery for later modelling. Box segmentation of objects that will be modelled without texture projection. Normalised quality of different pieces (via image filtering)

Development Skills Required	C++ programming (with OpenCV), Matlab programming, image editing skills
Technology Required	OpenCV library, C++ compiler, Matlab, Adobe Photoshop
Product Dependencies	CGA6

Product ID	CGA8
Product Title	Textures
Product Description	Textures after transformation, restoration and completion, ready for projection or plain texturing of objects.
Product Form	JPEG image
Requirements	Textures completed in “perceptual filling-in” way (p. 13). Perspective distortions removed via transformation matrix (p. 19). No obvious defects or deformities on textures for projection. Appropriate scale of textures for plain texturing. Seamlessness of textures required for tiling.
Development Skills Required	Matlab programming, image editing skills
Technology Required	Matlab, Adobe Photoshop
Product Dependencies	CGA7

Product ID	CGA9
Product Title	Moodboard
Product Description	Assembly of segmented pieces based on various topics (colour, shapes, materials, object categories) to get understanding of painting aesthetics.
Product Form	PSD image
Requirements	-
Development Skills Required	-
Technology Required	Adobe Photoshop
Product	CGA7

Dependencies	

Product ID	CGA10
Product Title	Saliency Map
Product Description	Map that emphasise to which image parts human attention will be focused.
Product Form	BMP image
Requirements	-
Development Skills Required	-
Technology Required	Matlab (Saliency Toolbox)
Product Dependencies	CGA6

Product ID	CGA11
Product Title	Layout Map
Product Description	Map with estimated positions and orientation of the objects from the painting.
Product Form	JPEG image
Requirements	Map contains all large objects from painting. Arrangement of the objects is derived from linear perspective analysis.
Development Skills Required	Geometry knowledge, image editing skills
Technology Required	Adobe Photoshop
Product Dependencies	CGA5

Product ID	CGA12
Product Title	Skybox
Product Description	Six images that together form seamless texture for the cube to represent outer distant world.
Product Form	Six square JPEG images

Requirements	No visible distortion at the seams of skybox. Whole skybox retains colour scheme and texture quality from original source. Maximum size per image 1MB.
Development Skills Required	Image editing skills
Technology Required	Adobe Photoshop
Product Dependencies	CGA2, CGA7

Product ID	CGA13
Product Title	Terrain
Product Description	Ground part of the environment that consists of plane and/or heightmap and/or mesh.
Product Form	JPEG image of terrain texture for plane / BMP image for heightmap generation + JPEG images of terrain textures / OBJ model with JPEG image of textures
Requirements	No visible boundaries in tiling (seamless). Regular hills within given aesthetics. No unnatural texture stretching for mesh models of terrain. No illogical terrain composition (e.g. lakes leaning).
Development Skills Required	Image editing skills, modelling skills, C++ programming (with OpenCV), OpenGL programming, GLSL programming
Technology Required	Adobe Photoshop, Blender, OpenCV library, C++ compiler, OpenGL library, GLSL library
Product Dependencies	CGA2, CGA8

Product ID	CGA14
Product Title	Meshes
Product Description	3D object models with their respective textures.
Product Form	OBJ model with JPEG image textures
Requirements	Low polygon meshes. Textures used for simulating geometry when possible. Texture baking used if necessary for plain texture.

	Polygon budget for individual mesh is weighted by its size in image and importance given by saliency map.
Development Skills Required	Modelling skills, image editing skills
Technology Required	Blender, Adobe Photoshop
Product Dependencies	CGA8

Product ID	CGA15
Product Title	Lighting
Product Description	Lighting for the whole scene defined by its position and colours of its three components (ambient, diffuse and specular).
Product Form	Code in application, code in shaders
Requirements	As small number of light sources as possible (ideally one). Neutral colour of light given by same value of each colour component. Lowered specular component.
Development Skills Required	OpenGL programming, GLSL programming
Technology Required	OpenGL library, GLSL library
Product Dependencies	CGA2, CGA12, CGA13, CGA14

Product ID	CGA16
Product Title	Camera
Product Description	Synthetic camera model that mimic behaviour of real camera and brings projected image of virtual 3D scene.
Product Form	Code in application
Requirements	Reasonable vertical viewing angle (between 20 and 50 degrees). Various camera modes and views including front, left and right view when in movement mode, top view in still mode and free movement mode. Far plane is far enough to comprise whole environment.
Development Skills Required	OpenGL programming

Technology Required	OpenGL library
Product Dependencies	CGA2, CGA11, CGA12

Product ID	CGA17
Product Title	Tour Track
Product Description	Predefined invisible track along which camera will move around the scene.
Product Form	Code in application
Requirements	Designed to go around all interesting assets of the scene. Moving in vertical dimension as well as in horizontal. Smooth turns and navigation without tugging.
Development Skills Required	OpenGL programming
Technology Required	OpenGL library
Product Dependencies	CGA2, CGA13, CGA14, CGA16

Product ID	CUG1
Product Title	Conversion User Guide
Product Description	Online material that provides guideline how to do painting conversion into 3D environment.
Product Form	Blog
Requirements	Simple content management system. Have discussion forums. Have post categories and search. Programmable CSS.
Development Skills Required	System administration, CSS programming
Technology Required	WordPress
Dependencies	CUG3, CUG4, CUG5

Product ID	CUG2
Product Title	Project Website
Product Description	Website for project outputs presentation, integrating blog and providing testing area.
Product Form	Website
Requirements	Responsive web design. Separate pages for presenting individual objects. Changing content on the home page. Contains testing area.
Development Skills Required	HTML programming, CSS programming, PHP programming, JavaScript programming, System administration
Technology Required	HTML, CSS, PHP, JavaScript, online hosting
Dependencies	CUG1, CUG3, CUG4, CUG5, CUG6, CUG7

Product ID	CUG3
Product Title	Project Logo
Product Description	Logo of the project website.
Product Form	PNG image
Requirements	Simple letters, simple graphics in project website colour scheme.
Development Skills Required	Image editing skills
Technology Required	Adobe Photoshop
Dependencies	CUG4

Product ID	CUG4
Product Title	Colour Scheme
Product Description	Colour scheme for the project website.
Product Form	PSD image
Requirements	Dark colour scheme. Colour hue limited to one.
Development Skills	-

Required	
Technology Required	Adobe Photoshop, Adobe Kuler
Dependencies	-

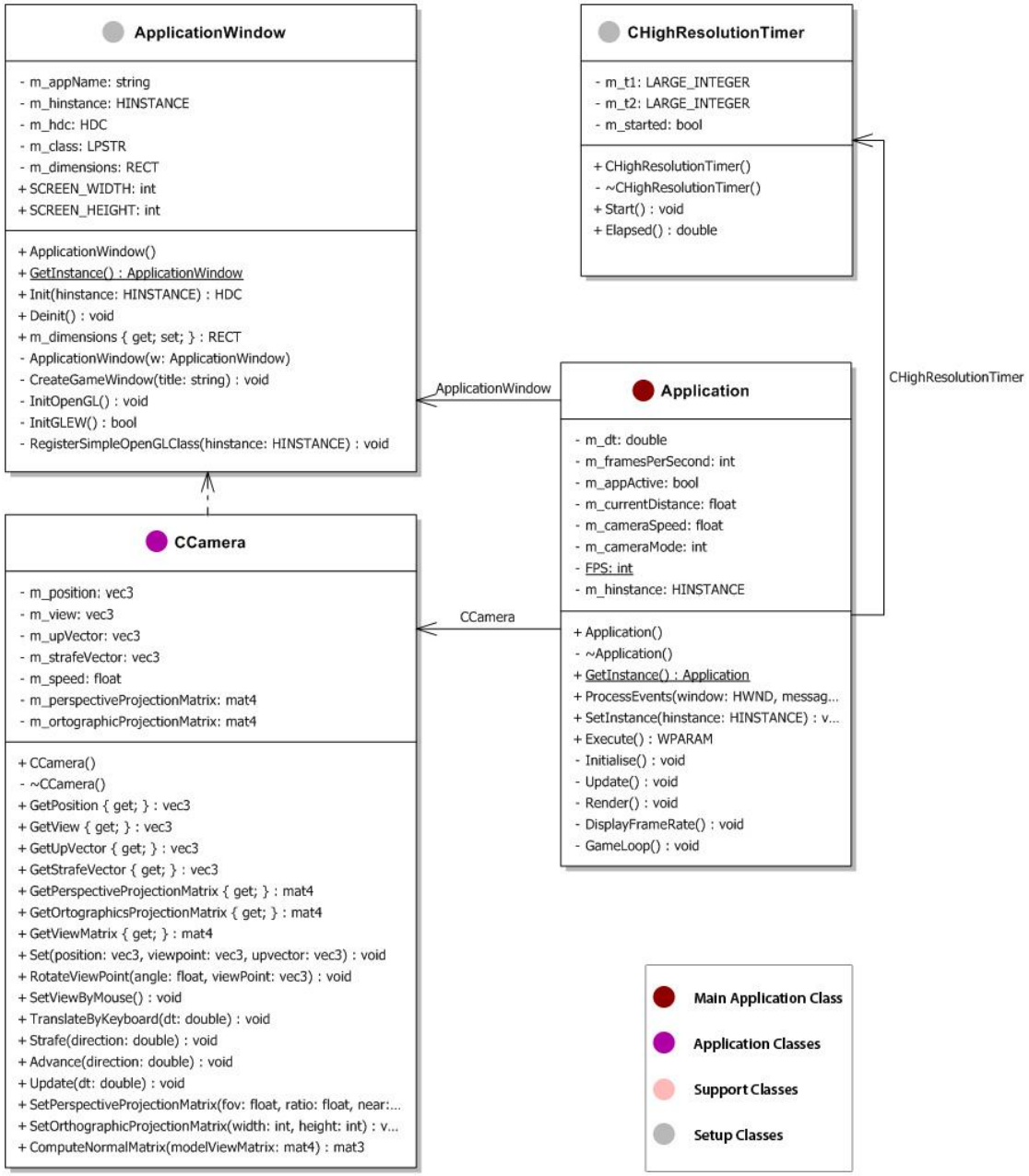
Product ID	CUG5
Product Title	Information Architecture
Product Description	Hierarchical organisation of the content of the website and appropriate labelling
Product Form	Table
Requirements	2-level information hierarchy. Menus with active sections highlighting. Breadcrumb navigation.
Development Skills Required	-
Technology Required	-
Dependencies	-

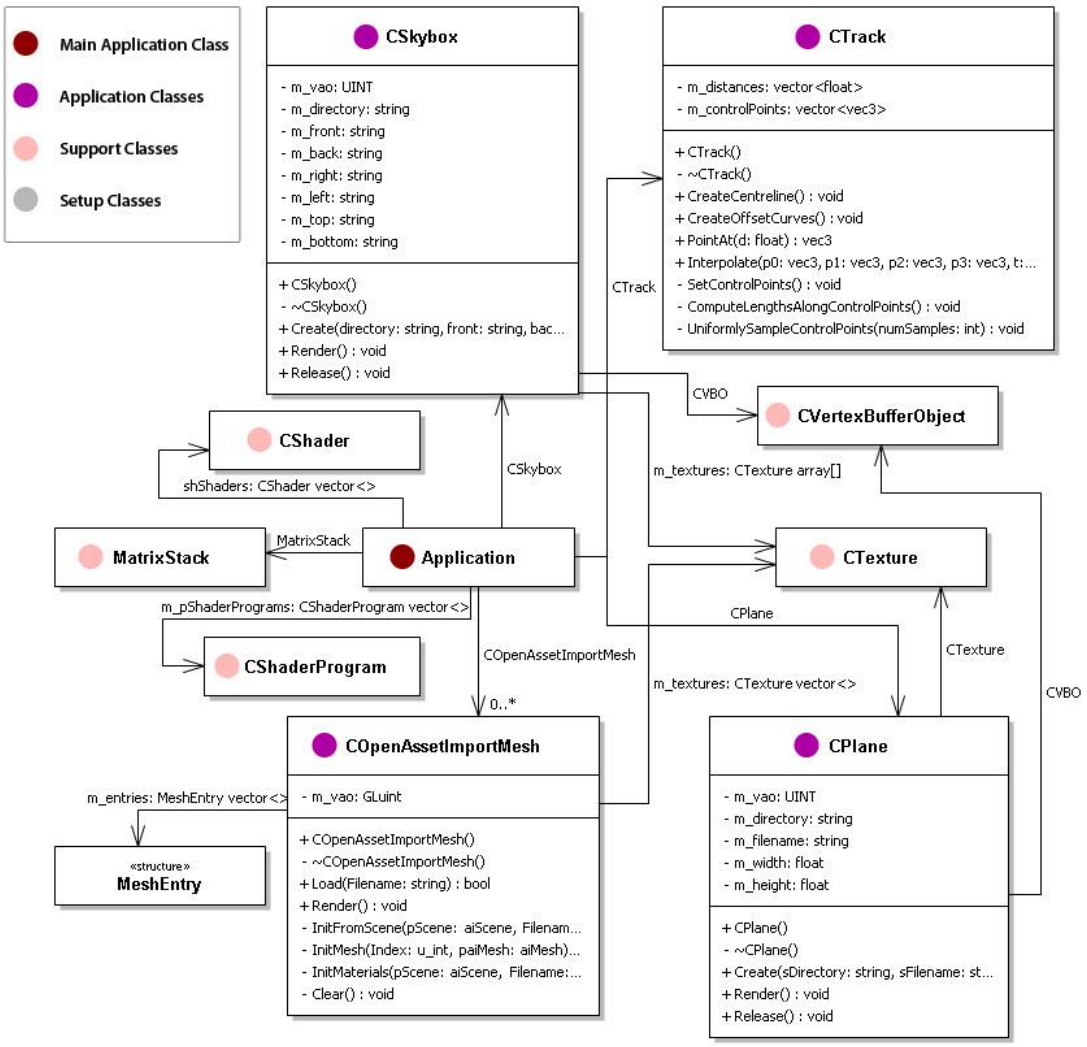
Product ID	CUG6
Product Title	Wireframe designs
Product Description	Schematic page layout blueprint.
Product Form	JPEG images
Requirements	Wireframe of home page for the biggest and smallest device.
Development Skills Required	-
Technology Required	Wireframe.cc
Dependencies	-

Product ID	CUG7
Product Title	Pages templates

Product Description	Template code for individual pages that simplifies creation and maintenance of the website.
Product Form	PHP files
Requirements	Different template for home page and other pages. Contains as much common code as possible.
Development Skills Required	PHP programming, HTML programming
Technology Required	PHP, HTML
Dependencies	CUG5

Appendix C – Class Diagram for Application




Main Application Class
Application Classes
Support Classes
Setup Classes
CSkybox
- m_vao: UINT
- m_directory: string
- m_front: string
- m_back: string
- m_right: string
- m_left: string
- m_top: string
- m_bottom: string
+ CSkybox()
- ~CSkybox()
+ Create(directory: string, front: string, bac...
+ Render() : void
+ Release() : void
CTrack
- m_distances: vector<float>
- m_controlPoints: vector<vec3>
+ CTrack()
- ~CTrack()
+ CreateCentreline() : void
+ CreateOffsetCurves() : void
+ PointAt(d: float) : vec3
+ Interpolate(p0: vec3, p1: vec3, p2: vec3, p3: vec3, t:...
- SetControlPoints() : void
- ComputeLengthsAlongControlPoints() : void
- UniformlySampleControlPoints(numSamples: int) : void
CTrack
CVBO
CVertexBufferObject
CShader
shShaders: CShader vector<>
CSkybox
m_textures: CTexture array[]
MatrixStack
MatrixStack
Application
CTexture
m_pShaderPrograms: CShaderProgram vector<>
CPlane
CShaderProgram
COpenAssetImportMesh
CTexture
m_textures: CTexture vector<>
CVBO
0..*
COpenAssetImportMesh
- m_vao: GLuint
+ COpenAssetImportMesh()
- ~COpenAssetImportMesh()
+ Load(Filename: string) : bool
+ Render() : void
- InitFromScene(pScene: aiScene, Filenam...
- InitMesh(Index: u_int, paiMesh: aiMesh)...
- InitMaterials(pScene: aiScene, Filename:...
- Clear() : void
CPlane
- m_vao: UINT
- m_directory: string
- m_filename: string
- m_width: float
- m_height: float
+ CPlane()
- ~CPlane()
+ Create(sDirectory: string, sFilename: st...
+ Render() : void
+ Release() : void
m_entries: MeshEntry vector<>
«structure»
MeshEntry

Appendix D – Algorithm for Colour Palette Generation

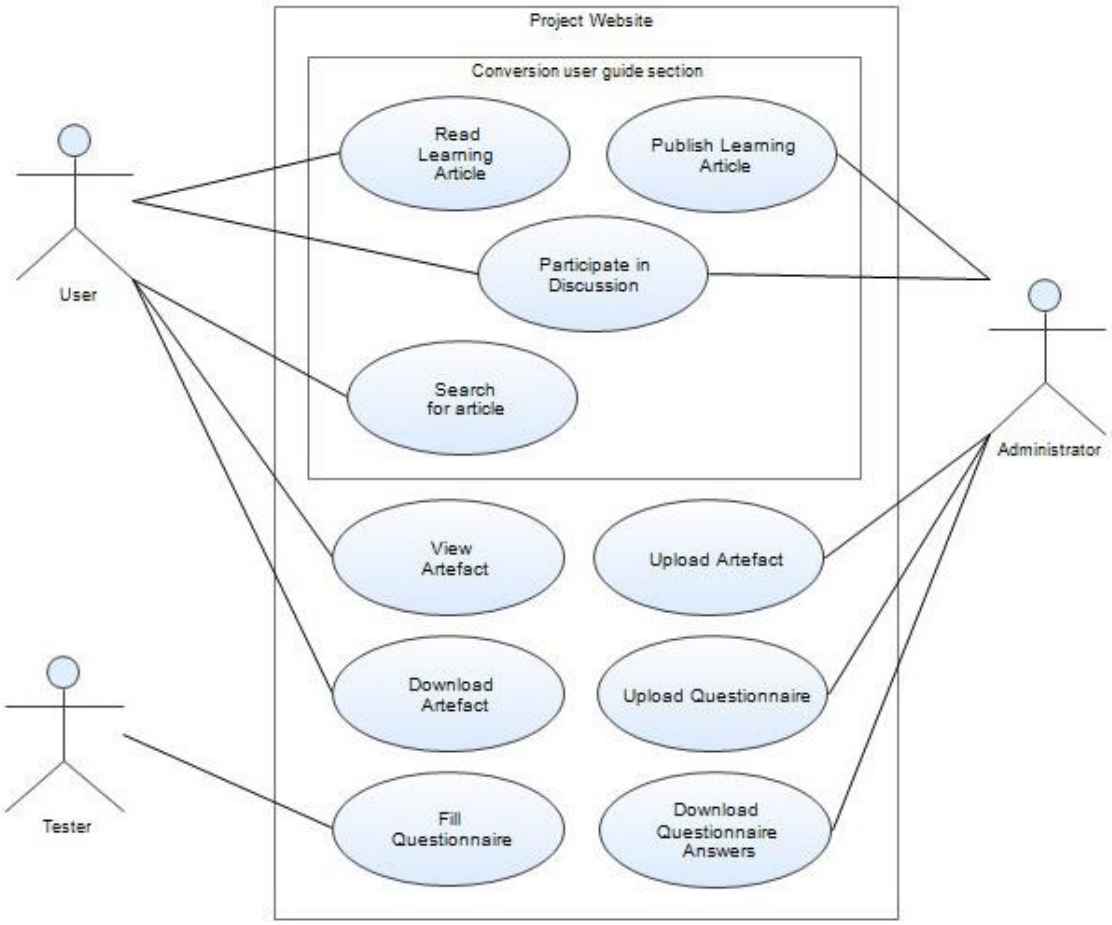
```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>
#include <sstream>
#include <algorithm>
#include <cmath>
#include <vector>
#include <unordered_map>

#define IMG_SIZE 1000

using namespace std;
using namespace cv;

//Structure defining colour group with number of pixels
struct CG {
   int code;
   int count;

   CG(int _code, int _count): code(_code), count(_count) {}
};

//Functions declaration
template<class T>
int chan_to_num(T blue, T green, T red);

vector<uchar> num_to_chan(int channels);

void draw_circle(Mat img, Point center, CG colour, double size);

inline bool compare_groups(CG colour1, CG colour2) {
   return (colour1.count > colour2.count);
}


//Main program
int main( int argc, char* argv[])
{
    // Check input parameters
    if (argc < 4)
    {
        cout << "Not enough parameters" << endl;
        return -1;
    }

   // Define matrices and read original image
    Mat I, J;
    I = imread(argv[1], CV_LOAD_IMAGE_COLOR);

   // Load of image failed
    if (!I.data)
    {
        cout << "The image" << argv[1] << " could not be loaded." << endl;
        return -1;
```

```
 }

// Read colour reduction parameter
 int colourReduction = 0;
 stringstream s;
 s << argv[2];
 s >> colourReduction;

// Error in input parameter for colour reduction
 if (!s || !colourReduction)
 {
     cout << "Invalid number entered for reducing colours. " << endl;
     return -1;
 }

// Read pixel number treshold parameter
 int pixelTreshold = 0;
 stringstream t;
 t << argv[3];
 t >> pixelTreshold;

// Error in input parameter for pixel treshold
 if (!t || !pixelTreshold)
 {
     cout << "Invalid number entered for limiting pixel groups. " << endl;
     return -1;
 }

// Calculate look-up table for colour set reduction
 uchar table[256];
int middleValue = colourReduction/2;

 for (int i = 0; i < 256; ++i)
    table[i] = (uchar)(colourReduction * (i/colourReduction) + middleValue);

 Mat lookUpTable(1, 256, CV_8U);
 uchar* p = lookUpTable.data;
 for( int i = 0; i < 256; ++i)
     p[i] = table[i];

// Function for look-up table transform of an image
 LUT(I, lookUpTable, J);

// Create image with reduced set of colours
imwrite("reduced.jpg", J);

// Create hash table and set initial value for each colour group as zero
unordered_map<int, int> colourGroups;
int singleGroup = 256/colourReduction;

uchar table2[256];
for (int i = 0; i < 256; ++i)
{
   table2[i] = i*colourReduction + middleValue;
}

for (int i = 0; i < singleGroup; ++i)
{
   int a = table2[i];

   for (int j = 0; j < singleGroup; ++j)
   {
```

```
            int b = table2[j];

            for (int k = 0; k < singleGroup; ++k)
            {
               int c = table2[k];

               int index = chan_to_num(a, b, c);
               colourGroups[index] = 0;
            }
         }
      }

      // Iterate over image to count numbers of pixels for each colour group
      Size matSize = J.size();
      for (int i = 0; i < matSize.height; ++i)
      {
         for (int j = 0; j < matSize.width; ++j)
         {
            Vec3b intensity = J.at<Vec3b>(i, j);
            int colourValue = chan_to_num(intensity.val[0], intensity.val[1],
intensity.val[2]);
            colourGroups[colourValue] += 1;

         }
      }

      //Create and populate container for final palette
      vector<CG> finalPalette;

      for (auto it = colourGroups.begin(); it != colourGroups.end(); ++it)
      {
         if (it->second >= pixelTreshold) {
            CG filter(it->first, it->second);
            finalPalette.push_back(filter);
         }
      }

      //Sort final palette with groups, starting with the largest
      sort(finalPalette.begin(), finalPalette.end(), compare_groups);

      //Find square space for each colour in the final palette
      int sqrSide = ((int)sqrt(finalPalette.size())+1);
      int ratio = IMG_SIZE/sqrSide;
      double maxSize = (double)(ratio-2)/(log((double)finalPalette[0].count));

      //Create final palette images with black and white backgrounds
      Mat paletteImageB = Mat::zeros( IMG_SIZE, IMG_SIZE, CV_8UC3 );
      Mat paletteImageW( IMG_SIZE, IMG_SIZE, CV_8UC3, Scalar(255, 255, 255) );
      int counter = 0;

      for  (int i = 0; i < sqrSide; ++i)
      {
         for (int j = 0; j < sqrSide; ++j) {
            draw_circle(paletteImageB, Point( ratio*(j+0.5), ratio*(i+0.5)),
finalPalette[counter], maxSize);
            draw_circle(paletteImageW, Point( ratio*(j+0.5), ratio*(i+0.5)),
finalPalette[counter], maxSize);
            ++counter;
            if(counter == finalPalette.size())
               break;
         }
```

```
      if(counter == finalPalette.size())
         break;
   }

   imwrite( "palette_black.jpg", paletteImageB);
   imwrite( "palette_white.jpg", paletteImageW);

    return 0;
}


//Function that transforms colour channels to one identifier that still carries
information about channels
template<class T>
int chan_to_num(T blue, T green, T red)
{
   int channels;

   channels = (int)blue*1000000 + (int)green*1000 + (int)red;

   return channels;
}

//Function that recovers colour channels from colour identifier
vector<uchar> num_to_chan(int channels)
{
   vector<uchar> individual;

   individual.push_back((uchar)(channels/1000000));
   individual.push_back((uchar)((channels%1000000)/1000));
   individual.push_back((uchar)(channels%1000));

   return individual;
}

//Draw circle function
void draw_circle(Mat img, Point center, CG colour, double size)
{
   int thickness = -1;
   int lineType = 8;
   vector<uchar> vec = num_to_chan(colour.code);

   circle( img, center, log((double)colour.count)*size/2, Scalar( vec[0], vec[1],
vec[2]), thickness, lineType);
}
```

Appendix E – Website Use Case Model



Appendix F – Website Templates

Header template (_header_pages.php)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title><?php echo $pageName; ?></title>
    <meta charset="utf-8">
    <meta name="description" content="Adding a Dimension: Turning a Famous Painting into a 3D Scene">
    <meta name="keywords" content="2D to 3D, 3D reconstruction, skybox, heightmap, seamless texture, 3D modelling, painting reconstruction, low-poly models">
    <meta name="rating" content="General">
    <meta name="robots" content="All">
    <meta name="revisit-after" content="7 days">
    <meta name="viewport" content="width=device-width; initial-scale=1.0">

    <link rel="stylesheet" href="/project/styles/basic.css"  type="text/css">
    <link rel="stylesheet" href="/project/styles/pages.css"  type="text/css">
    <link rel="home" href="http://www.virtualworld.sk/project">

    <script>src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
    <script>… Function for scrolling header…</script>
    <script>…Function for navigation appearance…</script>
</head>

<body>
    <header id = "Header">
        <div class = "headerContainer">
            <div class = "headerPanel">
                <div id = "Logo" class = "logo">
```

```
                <a href="http://www.virtualworld.sk/project"><img src =
                "/project/images/Logo.png" alt = "Logo"/></a>
            </div>
            <div id="Menu" class = "menu">
                <nav class = "menuContainer">
                    <a href="" id = "MenuIcon"><img src = "/project/images/menu.png" alt =
                    "Menu Icon"></a>
                    <ul id = "MenuList" class = "hiddenSmall">
                        <li><a href="/project">Home</a></li>
                        <li><a href="http://blog.virtualworld.sk">Blog</a></li>
                        <li><a href="/project/textures" <?php if($pageDirectory ==
                        'textures'){?> class="active"<?php }?>>Textures</a></li>
                        <li><a href="/project/models" <?php if($pageDirectory == 'models'){?>
                        class="active"<?php }?>>Models</a></li>
                        <li><a href="/project/algorithms" <?php if($pageDirectory ==
                        'algorithms'){?> class="active"<?php }?>>Algorithms</a></li>
                        <li><a href="/project/about" <?php if($pageDirectory == 'about'){?>
                        class="active"<?php }?>>About</a></li>
                    </ul>
                </nav>
            </div>
            <br class="clear">
        </div>
    </div>
</header>
<div id = "Section">
    <div class = "sectionContainer">
        <div class = "breadcrumb">
            <h1><?php echo $pageDirectory; ?></h1>
            <div class = "navigation">
                <?php
                    if($pageOn == 'index.php') {
                        echo "Home";
                    }
```

```
                    else {
                        echo '<a href = "/project/'.$pageDirectory.'">Home</a> / '.$pageName;
                    }
                ?>
            </div>
        </div>
```

Footer template (_footer_pages.php)

```
        </div>
    </div>
    <footer id = "Footer">
        <div class = "footerContainer">
            <p>&copy; 2015 virtualworld.sk</p>
        </div>
    </footer>
</body>
</html>
```

Appendix G – JavaScript Functions

Function for scrolling header

```
<script>
    $(document).ready(function(){
        $(window).scroll(function() {
            var height = $(window).scrollTop();
            if(height  > 0) {
                $("#Header").addClass("scrolled");
            }
            else {
                $("#Header").removeClass("scrolled");
            }
        });
    });
</script>
```

Function for navigation appearance

```
<script>
    $(document).ready(function(){
        $( "#MenuIcon" ).on( "click" , function(e) {
            e.preventDefault();
            var status = $('#MenuList').attr('class');
            if(status == "hiddenSmall") {
                $("#MenuList").removeClass("hiddenSmall");
                $("#MenuList").addClass("activeSmall");
            }
            else {
                $("#MenuList").removeClass("activeSmall");
                $("#MenuList").addClass("hiddenSmall");
```

```
            }
        });
});
</script>
```

Appendix H – Algorithm for Seamless Edges

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>
#include <sstream>
#include <algorithm>
#include <cmath>
#include <vector>
#include <unordered_map>

#define IMG_SIZE 1000

using namespace std;
using namespace cv;


int get_contrast(Mat &Image, int neighbourhood);

void create_overlay(const Mat &Original, Mat &New, int overlay);

void apply_overlay(Mat &Overlay, Mat &Final, int overlay);


int main( int argc, char* argv[])
{
    // Check input parameters
    if (argc < 3)
    {
        cout << "Not enough parameters" << endl;
        return -1;
    }

   // Define matrix and read image
    Mat I;
    I = imread(argv[1], CV_LOAD_IMAGE_COLOR);

   // Load of image failed
    if (!I.data)
    {
        cout << "The image " << argv[1] << " could not be loaded." << endl;
        return -1;
    }

   Size imageSize = I.size();

   //Texture is not square
   if (imageSize.height != imageSize.width)
   {
      cout << "The image" << argv[1] << " is not a square." << endl;
        return -1;
   }

   // Read border overlay parameter (btw 2 - size of the image)
   int overlay = 0;
    stringstream t;
    t << argv[3];
    t >> overlay;
```

```
   // Error in input parameter for border overlay
    if (!t || !overlay)
    {
        cout << "Invalid number entered for border overlay. " << endl;
        return -1;
    }

   if (overlay < 2)
   {
      cout << "Number entered for border overlay is too small. " << endl;
      return -1;
   }

   if (overlay > imageSize.height)
   {
      cout << "Number entered for border overlay is larger than the image size. "
<< endl;
      return -1;
   }

   //Calculate and display contrast for neighbourhood size of 3, 7, and 11
   int cont3 = get_contrast(I, 3);
   int cont7 = get_contrast(I, 7);
   int cont11 = get_contrast(I, 11);

   cout << "Contrast range mean for 3X3 area: " << cont3 << endl;
   cout << "Contrast range mean for 7X7 area: " << cont7 << endl;
   cout << "Contrast range mean for 11X11 area: " << cont11 << endl;

   //Create overlay image
   Mat O(imageSize.height, overlay, CV_8UC3, Scalar(0,0,0));

   create_overlay(I, O, overlay);

   //Create base for final image
   Mat F = I.clone();

   //Create base for overlay modification
   Mat D(imageSize.height, overlay, CV_8UC3, Scalar(0,0,0));

   //Show overlay image
   namedWindow( "Overlay", WINDOW_NORMAL );
   imshow( "Overlay", O );
   waitKey(1000);

   //Provide choice for further modification
   cout << "What filter do you want to apply? (n - none, g - gaussian, m -
median)" << endl;

   // Get the answer
   char toolchoice, apply;
   cin >> toolchoice;

   //Create base matrices for overlay modification
   Mat D1(imageSize.height, overlay, CV_8UC3, Scalar(0,0,0));
   Mat D2(imageSize.height, overlay, CV_8UC3, Scalar(0,0,0));
   Mat D3(imageSize.height, overlay, CV_8UC3, Scalar(0,0,0));
   Mat D4(imageSize.height, overlay, CV_8UC3, Scalar(0,0,0));
   Mat D5(imageSize.height, overlay, CV_8UC3, Scalar(0,0,0));

   //Apply filtering according to user choice
```

```
   switch(toolchoice) {
      case 'n':
         cout << "Apply overlay onto image? (y/n)" << endl;
         cin >> apply;
         if (apply == 'y' || apply == 'Y')
            apply_overlay(O, F, overlay);
         break;
      case 'm':
         namedWindow( "1", WINDOW_NORMAL);
         medianBlur ( O, D1, 3 );
         imshow( "1", D1 );
         waitKey(1000);
         namedWindow( "2", WINDOW_NORMAL);
         medianBlur ( O, D2, 5 );
         imshow( "2", D2 );
         waitKey(1000);
         namedWindow( "3", WINDOW_NORMAL);
         medianBlur ( O, D3, 7 );
         imshow( "3", D3 );
         waitKey(1000);
         namedWindow( "4", WINDOW_NORMAL);
         medianBlur ( O, D4, 9 );
         imshow( "4", D4 );
         waitKey(1000);
         namedWindow( "5", WINDOW_NORMAL);
         medianBlur ( O, D5, 11 );
         imshow( "5", D5 );
         waitKey(1000);
         cout << "Choose which image version you want to apply on original image
(1, 2, 3, 4, 5, n - none)" << endl;
         cin >> apply;
         switch(apply) {
            case '1':
               apply_overlay(D1, F, overlay);
               break;
            case '2':
               apply_overlay(D2, F, overlay);
               break;
            case '3':
               apply_overlay(D3, F, overlay);
               break;
            case '4':
               apply_overlay(D4, F, overlay);
               break;
            case '5':
               apply_overlay(D5, F, overlay);
               break;
         }
         break;
      case 'g':
         namedWindow( "1", WINDOW_NORMAL);
         GaussianBlur( O, D1, Size( 3, 3 ), 0, 0 );
         imshow( "1", D1 );
         waitKey(1000);
         namedWindow( "2", WINDOW_NORMAL);
         GaussianBlur( O, D2, Size( 5, 5 ), 0, 0 );
         imshow( "2", D2 );
         waitKey(1000);
         namedWindow( "3", WINDOW_NORMAL);
         GaussianBlur( O, D3, Size( 7, 7 ), 0, 0 );
         imshow( "3", D3 );
         waitKey(1000);
```

```
        namedWindow( "4", WINDOW_NORMAL);
        GaussianBlur( O, D4, Size( 9, 9 ), 0, 0 );
        imshow( "4", D3 );
        waitKey(1000);
        namedWindow( "5", WINDOW_NORMAL);
        GaussianBlur( O, D5, Size( 11, 11), 0, 0 );
        imshow( "5", D3 );
        waitKey(1000);
        cout << "Choose which image version you want to apply on original image
(1, 2, 3, 4, 5, n - none)" << endl;
        cin >> apply;
        switch(apply) {
           case '1':
              apply_overlay(D1, F, overlay);
              break;
           case '2':
              apply_overlay(D2, F, overlay);
              break;
           case '3':
              apply_overlay(D3, F, overlay);
              break;
           case '4':
              apply_overlay(D4, F, overlay);
              break;
           case '5':
              apply_overlay(D5, F, overlay);
              break;
        }
        break;
   }

    return 0;
}

//Calculate contrast function
int get_contrast(Mat &Image, int neighbourhood)
{

   Size imageSize = Image.size();

   int step, begin, end, newSize;
   step = neighbourhood/2;
   begin = step;
   end = imageSize.height-step;
   newSize = (imageSize.height - neighbourhood) + 1;

   long int countB = 0, countG = 0, countR = 0;

   Mat C(newSize, newSize, CV_8UC3, Scalar(0,0,0));

   for (int i = begin; i < end; ++i)
   {
      for (int j = begin; j < end; ++j) {

         Vec3b pixel = Image.at<Vec3b>(i, j);
         int maxB = 0, maxG = 0, maxR = 0;

         int ibegin, iend;
         ibegin = i-step;
         iend = i+step;

         for (int k = ibegin; k <= iend; ++k)
```

```
        {
           int jbegin, jend;
           jbegin = j-step;
           jend = j+step;

           for (int l = jbegin; l <= jend; ++l)
           {

              Vec3b neighbour = Image.at<Vec3b>(k,l);

              int differenceB = abs(pixel[0] - neighbour[0]);
              if (differenceB > maxB)
                 maxB = differenceB;

              int differenceG = abs(pixel[1] - neighbour[1]);
              if (differenceG > maxG)
                 maxG = differenceG;

              int differenceR = abs(pixel[2] - neighbour[2]);
              if (differenceR > maxR)
                 maxR = differenceR;

           }
        }

        C.at<Vec3b>((i-step), (j-step)).val[0] = maxB;
        C.at<Vec3b>((i-step), (j-step)).val[1] = maxG;
        C.at<Vec3b>((i-step), (j-step)).val[2] = maxR;

        countB += maxB;
        countG += maxG;
        countR += maxR;

     }
  }

  int mean = (countB/pow(newSize,2) + countG/pow(newSize,2) +
countR/pow(newSize,2))/3;

  return mean;

}

//Create overlay function
void create_overlay(const Mat &Original, Mat &New, int overlay)
{
  Size imageSize = Original.size();

  Mat M(imageSize.height, overlay, CV_8U);

  for (int i = 0; i < overlay; ++i)
  {
     for (int j = 0; j < imageSize.height; ++j)
     {
        M.at<uchar>(j, i) = (uchar)(((double) rand() / (RAND_MAX)) +
((1.0/overlay))*(overlay - (i+1)));

        if ((int)M.at<uchar>(j, i) == 0)
        {
           New.at<Vec3b>(j, i)[0] = Original.at<Vec3b>(j, i)[0];
           New.at<Vec3b>(j, i)[1] = Original.at<Vec3b>(j, i)[1];
           New.at<Vec3b>(j, i)[2] = Original.at<Vec3b>(j, i)[2];
```

```
            }
            else
            {
               New.at<Vec3b>(j, i)[0] = Original.at<Vec3b>(j, ((imageSize.width - 1)
- i))[0];
               New.at<Vec3b>(j, i)[1] = Original.at<Vec3b>(j, ((imageSize.width - 1)
- i))[1];
               New.at<Vec3b>(j, i)[2] = Original.at<Vec3b>(j, ((imageSize.width - 1)
- i))[2];
            }
         }
      }
   }

//Apply overlay onto image
void apply_overlay(Mat &Overlay, Mat &Final, int overlay)
{
   Size imageSize = Final.size();

   for (int i = 0; i < imageSize.height; ++i)
   {
      for (int j = 0; j < overlay; ++j)
      {
         Final.at<Vec3b>(i, j) = Overlay.at<Vec3b>(i, j);
      }
   }

   imwrite( "modified_image.jpg", Final);
}
```

Appendix I – Questionnaire

1. Considering the video and painting, do you think the colours of the scene in the video in comparison with painting are:
    a. Very accurate
    b. Rather accurate
    c. Rather inaccurate
    d. Very inaccurate

2. Considering the video and painting, do you think layout of the objects in the scene in the video in comparison with painting is:
    a. Very accurate
    b. Rather accurate
    c. Rather inaccurate
    d. Very inaccurate

3. When you watch the scene on video, do you perceive the environment that is not visible on painting as:
    a. Well fitted, without looking at painting I cannot tell where exactly the painting ends and rest of the scene begins
    b. Plausible, but there are few indicators making clear to me where the environment from painting ends
    c. Doesn't fit at all, reconstructed environment doesn't integrates with environment in painting

4. Considering the video and painting, do you think the shapes of objects (e.g. buildings, people, trees) in the video in comparison with painting are:
    a. Very accurate
    b. Rather accurate
    c. Rather inaccurate
    d. Very inaccurate

5. Considering the video and painting, do you think the sizes of objects (e.g. building, people, trees) in the video in comparison with painting are:
    a. Very accurate
    b. Rather accurate
    c. Rather inaccurate
    d. Very inaccurate

6. Would you be interested to have virtual tours around more paintings?
    a. Yes

b. No
c. I don’t know

Please tell us, if you have a preference for any painting.

7. Would you find it useful to have a guide telling you information about painting, when you having virtual tour?
    a. Yes
    b. No
    c. I don’t know

8. Would you like to play a computer game in the environment designed according to painting?
    a. Yes
    b. No
    c. I don’t know

9. Your age group is:
    a. 18-24
    b. 25-34
    c. 35-44
    d. 45-54
    e. 55+

10. Your email is:

Appendix J – Virtual Tour Track Control Points

```
m_controlPoints.push_back(glm::vec3(10, 12, -280));
m_controlPoints.push_back(glm::vec3(27, 2, -200));
m_controlPoints.push_back(glm::vec3(30, 1, -130));
m_controlPoints.push_back(glm::vec3(0, 2, -50));
m_controlPoints.push_back(glm::vec3(30, 7, 50));
m_controlPoints.push_back(glm::vec3(70, 8, 150));
m_controlPoints.push_back(glm::vec3(40, 8, 250));
m_controlPoints.push_back(glm::vec3(-10, 8, 300));
m_controlPoints.push_back(glm::vec3(-90, 8, 270));
m_controlPoints.push_back(glm::vec3(-70, 8, 150));
m_controlPoints.push_back(glm::vec3(-35, 6, 0));
m_controlPoints.push_back(glm::vec3(-47, 1, -150));
m_controlPoints.push_back(glm::vec3(0, 0, -190));
m_controlPoints.push_back(glm::vec3( 47, 1, -150));
m_controlPoints.push_back(glm::vec3(30, 6, 0));
m_controlPoints.push_back(glm::vec3(-40, 8, 0));
m_controlPoints.push_back(glm::vec3(-70, 8, -100));
m_controlPoints.push_back(glm::vec3(-160, 8, -250));
m_controlPoints.push_back(glm::vec3(-60, 8, -330));
m_controlPoints.push_back(glm::vec3(-40, 8, -390));
m_controlPoints.push_back(glm::vec3(-0, 8, -420));
m_controlPoints.push_back(glm::vec3(35, 8, -400));
m_controlPoints.push_back(glm::vec3(50, 8, -350));
```

Appendix K – List of Files Submitted on USB

- Algorithms
- Application
- Application video
- Models
- Project report
- Raw questionnaire data
- Website