UNIVERSITY OF GOTHENBURG

Shotcrete Simulator

For Education of Shotcrete Robot Operators

Master of Science Thesis in Computer Science

Petter Börjesson,

Mattias Thell

University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, October 2009

Shotcrete Simulator
For Education of Shotcrete Robot Operators

Petter Börjesson,
Mattias Thell

Examiner: Ulf Assarsson

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden September 2009

Abstract

Sprayed concrete, or shotcrete, is a commonly used technique for structural reinforcement. Until now, there has been no effort in creating a virtual environment to educate personnel for this kind of work. This thesis presents the work done in implementing a prototype for a virtual simulation environment to be used in the education of personnel for shotcrete application. The prototype features an environment in which the user can make use of shotcrete robot equipment to apply concrete to surfaces. It also features several statistical tools for evaluation purposes. Since the final program cannnot be considered a complete product, enhancements that need to be implemented to achieve such a goal is also discussed.

Sammanfattning

Sprutbetong är en vanlig metod för förstärkning av till exempel tunnelbyggen. Fram till nu har ingen insats gjorts för att skapa en virtuell miljö för att utbilda personal för denna typ av arbete. Den här uppsatsen presenterar arbetet som gjorts för att skapa en prototyp för en virtuell träningsmiljö. Prototypen innefattar en miljö i vilken användaren kan styra en sprutbetongrobot för att applicera betong på ytor. Den innefattar även diverse verktyg för att evaluera resultatet. Eftersom denna versionen av simulatorn inte kan ses som komplett, disukuteras även fortsatt utveckling.

Preface

This is a master thesis at Gothenburg University and Chalmers University of Technology, department of Computer Science and Engineering. This work was done from February 2009 to June 2009. The thesis was requested from and done in collaboration with the Swedish company BESAB. Mikael Johansson at the department of Construction and Engineering supervised the project and the examiner at Chalmers University of Technology was Ulf Assarsson.

All the work done in for this thesis was performed by Petter Börjesson and Mattias Thell. The work was done as a team and although small tasks were done individually all major parts of the thesis was done in collaboration.

Contact information:

Petter Börjesson:	petter.borjesson@gmail.com
Mattias Thell:	mattias.thell@gmail.com

We would like to thank BESAB and especially Tommy Ellison who gave us this opportunity and helped us during the project. We would also like to thank Mikael Johansson, Börje Westerdal and Mattias Roupé at the department of Construction Engineering at Chalmers University of Technology for their help during the work on this thesis.

Table of Contents

1 Introduction

This master's thesis will describe the development of a first prototype of a simulation environment for sprayed concrete reinforcement. Also known as shotcrete, this is a commonly used technique that is applied to tunnel walls and other structural elements. The need for a simulation environment has grown as the costs of educating personnel are very high. The final goal of such a simulation is to complement real life training of personnel with a virtual environment in a time and cost efficient way.

1.1 Background

In the field of construction engineering, an important part is the ability to rapidly and effectively strengthen existing structural elements using some kind of reinforcing material. Typically, the material which is used for this is concrete. The primary method for applying this concrete is to pneumatically project it onto a surface at high velocity. This method is called shotcrete reinforcement, or simply shotcrete.

When applying concrete this way, it can be used to strengthen many kinds of surfaces, including mountain faces, tunnel walls and even overhang areas like ceilings. Given this applicability, shotcrete reinforcement is used in many kinds of construction work counting transportation tunnels, mining operations, silos and bridges as some application areas. Shotcrete can also be used for repair work and restoration.

Historically, shotcrete has been applied by skilled professionals operating a hand-held hose. In order to maximize the efficiency of the procedure, the hose nozzle must be held in a precise manner, in ideal conditions at a certain distance, perpendicularly to the surface, for a desired amount of concrete to stick to the surface. Failure to meet the ideal conditions can result in greater amounts of rebound. That is, concrete which does not stick to the surface but bounces back and is wasted. Some rebound is unavoidable, but minimizing the amount of wasted concrete is important as a lot of waste quickly becomes expensive. Typically, there are also prescriptions regarding desired thickness and other properties on the applied concrete that need to be fulfilled.

In recent years however, the hand-held hoses have been gradually phased out in favor of robotically controlled variants (1). The introduction of robotic gear brought with it a number of benefits. Not only were health and safety of the operators improved but the equipment also reduced costs as the work could be aided by computers (1). While the robot mechanics greatly improves the working conditions for the operators, manual control of robots are still necessary to obtain satisfactory results in certain situations.

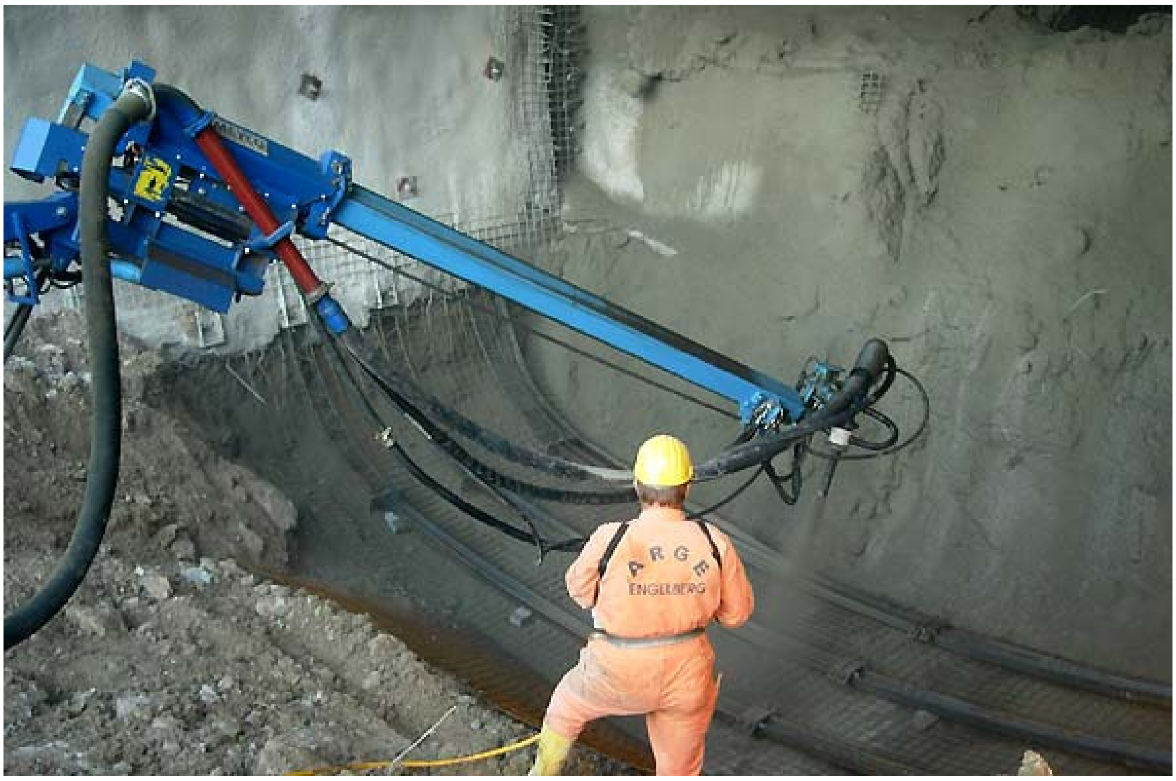


Figure 1: Shotcrete robot in action. (Source: Meyco)

For the technology to be used effectively, education is needed, and it is estimated that the needs for educated personnel will increase in the coming years (1). Currently in Sweden, there are no government funded educational entities that teach shotcrete reinforcement and there are no set standards for certification of robot operators. This places the responsibility of educating operators at individual companies who require personnel. The education is often associated with great costs and the schooling of a single shotcrete operator can reach sums of up to one million SEK (1). The education of an operator is usually divided into a number of parts with extensive practical training as the primary method of learning. This field training is most often performed together with an instructor with good practical knowledge of shotcrete reinforcement at a real working site. The major factors for the cost of the education are technical equipment and training on real work sites. Inevitably, students make mistakes while learning to handle the equipment. Mistakes must be corrected, either by applying further layers, or in the worst case, removal of a much too thick layer of concrete. In either case a lot of concrete is wasted. This together with higher than normal rebound in the beginning combined with possible delays at work sites make training very costly.

1.2 Previous Work

Before this project started, a pre-study on the problem of shotcrete simulations was written by Westerdahl et. Al, 2007 (1). This report describes the underlying problem of shotcrete and why a digital training simulator would be of great use for the industry. The authors also report on how shotcrete works and what parameters that affect the result. The report also describes the different kinds of shotcrete robot equipment that exist and how they work. Finally they give some basic ideas on how one could go about implementing a simulator for this purpose and how to computationally visualize the different components that are needed.

As shotcrete is widely used there has been research on how to calculate shotcrete behavior so that the result of spraying a concrete mixture on a certain surface can be predicted. This is very useful as workers can use these calculations to produce the most efficient concrete mixture for a specific project (2). One method used to perform these calculations is called Distinct Element Modeling (DEM) which has been proven to be able to calculate the behavior of shotcrete very accurately. Unfortunately, DEM is computationally heavy which makes it unsuitable for use in a real-time simulation environment (1).

There are many fields in the industry of today that uses simulators to educate personnel for various tasks. For example, simulators are used extensively to train aircraft pilots. As was shown in the pre study, it should definitely be possible to construct a simulator for training of people working with shotcrete. According to BESAB (3) representative Tommy Ellison, there has been surprisingly little work on the area of training simulations for shotcrete personnel. Indeed, searches made for products of this capacity have not yielded any result. This makes real time shotcrete simulations with focus on training robot operators a fairly new field. As was found during the production of this thesis, no research on this particular area has been conducted nor do such simulators exist.

2 Problem Specification

In the light of costly education and the increasing need for personnel, industrial needs necessitate spending large amounts of money to meet the demand for skilled shotcrete robot operators. The problem of expensive training could be alleviated by looking at virtual training environments to make the education of personnel more effective. A computer simulation could be constructed to provide a visual, as well as a physical, simulation of the shotcrete process with the goal of educating shotcrete operators. The use of a computer program like this could serve as a complement to the education and could greatly reduce costs and enable education of operators in an effective manner. Students operating a computer simulation would not induce costs of concrete removal or delays on real work sites. Also, many of the dangers and health risks associated with operating of robots at live construction sites could be eliminated. This thesis covers the creation and analysis of a prototype of such a program.

The pre study highlights important domain properties that should be taken into consideration if and when a simulation environment is built. The goals of this thesis are to expand on the ideas put forth in the pre study and build the foundation of a proper simulation program. That is, this thesis will not necessarily produce a fully operational educational simulation with all the aspects of such a program, but rather lay the ground work and ascertain the likelihood that a shotcrete simulator is indeed possible. A prototype, consisting of working systems for all of the more important parts, should be produced. To this end, it is necessary to find proper rendering and simulation techniques to visualize concrete and the environment drivers operate in.

There are several problems that need to be dealt with in order for the simulation environment to look and feel realistic. Since the simulator is intended to be used for educational purposes, one highly important part is that of realism. Realism, both in terms of visual and physical correctness, can be the key factor in determining whether or not an operator can use the system for training purposes and obtain satisfactory results. The visual representation must correlate with that which the operator expects to find in real life and the physical calculations must also be good enough so the simulation appears to be realistic.

This thesis will consider shotcrete reinforcement using robot mounted hoses only. The hand-held variants, though physically identical in terms of shotcrete adhesion and behavior, feature a couple of subtle but important differences. Robot mounted shotcrete equipment displaces the control of the nozzle from the hands of a human to the end of a robotic arm which is operated by a control device maneuvered by the operator. A basic such control device features various buttons and joysticks which enable the operator to control the robot with great accuracy. The fact that the equipment is not used by hand, but rather with a control device, fits the virtual environment much better than a hand-held system. The representation of a hand-held device would put great constraints on the virtual simulation. These constraints would likely be difficult, or outright impossible, to solve with the technology of today.

The most significant distinction is that directly operated equipment would need some kind of representation in the virtual world, making a direct mapping of control very hard. It would be necessary to implement some kind of interface between the two worlds which would decrease the realism of the interaction. When using robots however, which already features an interface control device, the interaction with the equipment can be made in much the same way as it would in a real environment.

If a simulation for shotcrete is to be constructed there are a few problems that need to be solved which will be the focus of this thesis:

Concrete visualization

Concrete appear in two forms in the context of shotcrete application. These two forms are:
- Fluid state, which is the wet mixture of concrete sprayed from the nozzle of the robot.
- Solid state, which is stuck or hardened concrete that have been applied to a surface.

Both forms need to be visualized in the simulation and different approaches are needed in order to obtain satisfactory results.

Adhesion calculations

When shotcrete hits a surface, some of it should stick to the surface and some of it might be rebounded. The amount that sticks depends on a large number of factors, such as angle and the distance from the surface. Calculations for these actions need to be computed and taken into account in the simulation.

Concrete data

The program needs to keep track of the result of concrete that has been used as well as the amount that has stuck to some surface. Firstly, so that it is possible to visualize the concrete during simulations. Secondly, it will also be used in order to be able to measure the result and gather various statistics about the shotcrete process. This is needed for quality assessment which could be considered one of the most important parts in a training simulation.

Operator control

Since the system is designed for a single desktop computer, a person cannot actually move around through the environment like she would in real life. So, except for controlling the shotcrete robot it must also be possible to control the virtual representation of the person operating the robot.

2.1 Limitations

During this project there will not be time to complete a fully fledged simulator ready for commercial use. The primary objective of the thesis is to establish if it is indeed possible to use virtual simulation to train operators of shotcrete robots. This means that the visualization and graphical representation of the environment and the objects in it will be the most important part and that certain limitations apply to the project.

The physical properties of concrete and how shotcrete behaves when hitting a surface, while important in some aspects of the simulation, are not something that will be a priority. Instead of doing a truly correct physical calculation, an approximation will be used at first and expanded upon if time allows it. Even though this part of the simulator is not a priority it should be easy to switch to a more correct simulation in the future.

During development of the program the focus will be to use it on a single screen Personal Computer and considerations for future use in environments like CAVE (4) or virtual reality systems will not be taken into account. That being said, potential future work will certainly allow such features to be built and it is indeed probable, or desirable, that such facilities exist if and when a fully operational educational environment is constructed.

The program aims to simulate the working conditions in a construction environment, and as such, should provide an audial as well as a visual experience. However, graphical visualization and representation will be the first priority. Development of audio systems will not be considered in this thesis.

There are a few different manufacturers of shotcrete robots on the market today and they each have a few different models available. In this first prototype a simple model will be used and there will be no focus on trying to correctly simulate all different kinds of robot models.

The virtual environment itself will need to be based on some real world reference. In real situations shotcrete can be used in many different environments. Simulating different scenarios will not be considered in this project but instead a mountain tunnel will used as an environment. This is a common setting for application of shotcrete and should suit the purposes of the application nicely. Also, since a tunnel environment rarely has precisely flat or orthogonal walls, and features many surfaces that could be considered difficult, such as ceilings and deformed rock walls, it should provide a sufficiently diverse testing environment.

The program is meant to be aimed at educating personnel in shotcrete reinforcement, and as such must feature certain requirements on the usability of the program. This means that the program will need menus and GUI components to handle all the functionality of the program in an easy to use manner for people with little or no computer background or education. These requirements will not be a priority during the development of the prototype but will be considered.

3 Technical Requirements

To get an understanding of what is needed to realize the simulator described in the problem specification an analysis was done from a technical point of view.

3.1 Understanding the problem

As stated previously, the goal of this project is foremost to ascertain the possibility to create a simulator for shotcrete robot drivers. If such a simulator should be of any use the most important thing is that the simulation both looks realistic enough and behaves as one would expect things to behave in real life. To be able to fulfill these goals and perform an analysis of the problem, a good understanding of how the shotcrete process works and what a working environment looks like is needed. Reference material in the form of videos and pictures has been found which will be the primary source of information in this respect. Also, experienced contact persons within the industry will be another valuable source of information regarding all aspects of shotcrete reinforcement.

3.2 Simulator parts

If the goals of the project are to be fulfilled there are a few questions that need to be answered from a technical point of view. What parts are needed for the system? How will they work and interact? What data is needed and how should it be saved?

3.2.1 Storing of concrete data

Shotcrete reinforcement works by spraying a target surface with concrete. The area on this surface will accumulate concrete and the thickness of the concrete layer will grow. Exactly how much concrete that has been applied on a specific area is important and so this information needs to be saved in the system in some way. This information needs to be available at each point of the surfaces of the simulated environment. This is needed both to be able to correctly visualize the concrete as well as performing quality measurements. The iterative way of work during shotcrete application means that it is not sufficient to only save information about whether or not a surface has shotcrete applied to it. It is also important that one can determine exactly how much concrete has been applied to each point of a surface depending on the output from a shotcrete robot.

The question about concrete data also leads to the question of how the system represents geometry, in this case in the form of a tunnel. Some mapping between data and geometry is needed as the geometry needs to know how much concrete is applied to each point on its surface. How concrete data is saved and how it is mapped to the geometry in the simulation will be one of the central parts of the system. Both the geometry of the environment and the concrete data need to be saved in a manner so that it is possible to render the data in a correct way. It should also be possible to use this data for statistical calculations during evaluation of the shotcrete process.

3.2.2 Visualization of concrete

As stated in the problem specification, concrete can exist in one of two states during shotcrete simulation. Either it is fluid in the form of concrete mixture while it is being shot onto the walls of a tunnel or it is solid when it has hardened on tunnel surfaces. There is a distinct difference between these states and the system will require different approaches to visualize them. When choosing rendering method for the concrete and designing the system parts that will handle the rendering it is important to think about how concrete behaves in these different states.

The rendering of solid concrete on walls will be tightly linked to the tunnel geometry and how concrete data is stored. When looking at the result from shotcrete work on real work sites one can notice that shotcrete tends to create a quite even surface with the occasional rounded shapes, similar to the rolling hills of a landscape. To accurately render a realistic concrete surface in the simulation the rendering method need to be able to give a realistic representation of actual depth in the concrete surface.

Rendering of the fluid state of the shotcrete ray is another problem. A shotcrete ray consists of many fast moving, small concrete particles mixed with water and chemicals. This makes the ray slightly transparent at the edges but opaque in the middle because of the amount of particles. In contrast to solid surfaces, these kinds of fuzzy, transparent objects need a different set of requirements and some method to represent a shotcrete ray need to be found. It is an important part for the simulation to look realistic and a very common method in computer graphics to represent these kinds of objects is to use particle systems (5).

3.2.3 Adhesion calculations

Adhesion, that is the ratio of concrete that is stuck on the surface and that which rebounds, also needs to be calculated. This can be divided in to two problems. First there is the problem to correctly calculate how much of the used concrete sticks to tunnel surfaces depending on variables such as angle to the surface and the velocity of the concrete. These calculations will use a simple system to approximate the result. The design of the system parts still has to be considered carefully so that the design allows for future replacement of the approximate method to something more correct if it is needed. Secondly, after the adhesion ratio has been calculated, the system must also take into consideration how this concrete is applied to the surfaces of a tunnel. In most cases the tunnel walls will not be smooth or even as the walls are often cut out of rock with explosives. During shotcrete process these small holes and cracks will be filled with concrete which will result in a smooth surface. Because of this, concrete cannot be applied evenly across a surface where the shotcrete ray hits because the roughness of the surface need also be considered.

3.2.4 Controlling the operator avatar and robot

In order to simulate a shotcrete robot, some system to handle the movement and control of the different parts of a robot must be designed. The system also needs to be able to receive input signals from devices like a keyboard, mouse or another device which replicates the behavior of robotic equipment. This must be connected to the simulation so that an operator can control the robot in a natural manner in as close accordance to real life as possible.

In a real work environment the operator of shotcrete robots can walk around the site with the remote control in hand, switching views by simply turning her head. This direct control over the field of vision is not possible in a simulation on a simple personal computer. Some way to control the camera that represents the operator's position must be devised.

3.2.5 Quality assessment

An important aspect of a training simulation is that the trainee should be able to get feedback on how well she performed. The system need to be able to display various statistical reports regarding the shotcrete process. For example, how much concrete was used, how much concrete was wasted, how much concrete has stuck to different parts of the tunnel surfaces, how thick the concrete layer is in different parts of the tunnel, etc. All this information must be visualized either in text or in some direct display on the walls so that the quality of the work can be evaluated.

4 Implementation

This chapter details the actual implementation of the various parts in the system. Choices and compromises of different techniques are also described, as well as design choices made during the implementation.

4.1 Programming Languages and Libraries

It was decided that, in order to accelerate the production, a higher level graphics library be used. The library that was decided upon was Open Scene Graph (OSG) (6), an open source solution which utilizes OpenGL (7) as the underlying API. Since OSG is written in C++, this was the logical choice in which to write the bulk of the application code. Vertex and fragment shaders are written in the OpenGL Shading Language (GLSL) (8).

4.2 Concrete Rendering

One of the major issues with a simulator in this domain is to determine which method of visualization should be used to render the concrete on the walls of a tunnel. It is important that this looks as realistic as possible so that drivers being educated in the simulator get a feeling of immersion and that the environment is sufficiently close to real world scenarios to provide a good learning experience. There exist many techniques to render surfaces in computer graphics and during the project several of these were tested in an attempt to figure out what works well for this scenario and what does not.

4.2.1 Multi-texturing

Texture mapping (9) is a basic concept in computer graphics and one way of visualizing concrete would be to use multiple textures during rendering. One possibility would be to use one texture for the stone surface and another texture for the concrete. Using a programmable fragment shader, one would then simply choose the type of texture which is suitable a particular time, depending on the surface information. A blend between the textures can be used to create a smooth transition from an area sprayed with concrete and one area without.

This technique is very simple to implement and it does not require much in terms of computational power. On the other hand, it suffers from some serious problems that prevent it from being very useful in the simulator presented here. The major problem being that even if lighting is computed for these kinds of surfaces they will, inherently, still look flat. When concrete is sprayed on a spot on the wall this spot should become increasingly extruded from the wall the more layers of concrete that are applied and this effect is simply not possible to simulate with multi-texturing alone. Fortunately, there are many ways to address this issue.

4.2.2 Normal mapping

One way to simulate unevenness of surfaces is to use normal mapping (10). This technique is very similar to regular texture mapping; the difference is that light is computed using additional normal information (which is usually stored in a texture of its own). Using this additional normal information during lighting calculations, this can make a surface appear uneven.

This technique is a big improvement from multi-texturing as sprayed concrete is not a perfectly even surface. Small bumps and extrusions in the surface can be simulated with normal maps. On graphical hardware of today, it is also very cheap and provides a good way of simulating coarseness, if not large dents and extrusions. To a small extent, it can also be used to make the concrete appear to be lying on top of the stone walls but there are still no "real" bumps on the surfaces which becomes very visible when looking at walls at a steep angle. This means that this is still not good enough to truly visualize sprayed concrete. Some way of visualizing concrete accumulation on the walls is needed to get a result realistic enough.

4.2.3 Vertex displacement mapping

A potentially good way of getting a good feeling that concrete is actually building up mass on the walls would be to use this technique. Instead of operating on a texture and lighting basis, this technique uses the underlying geometry to achieve uneven surfaces. Vertex displacement is executed during a vertex shader pass and can be used to shift vertices in some way, thus deforming geometry (11) (12). This is done by reading a value in the vertex shader, usually from a height map texture and modifying the vertex position according to this value. This modification could be to shift the vertex position along the vertex normal or one of the vertex components along one of the axis. It is obviously necessary to also shade the surface of the geometry and add textures. As a complement to displacement mapping, normal mapping would likely be used to achieve sufficiently realistic results.

Vertex displacement has a major advantage over the previous techniques as it actually deforms the geometry. Because of this, sprayed concrete could actually extrude from the original surface. Also, since this technique is shifting vertex positions instead of operating on just the pixel colors, it never produces any artifacts due to steep viewing angles which can be a problem with other techniques. There are however other problems with this technique. First of all, every surface would need to consist of a large amount of tightly packed vertices which means that the geometric complexity is high. Even with a very high number of vertices, it is hard to get a surface that is smooth enough to simulate concrete in the way that is needed. Secondly, vertex displacement can be difficult in certain situations. For example, when the original surface is flat, all vertices lie in the same plane and displacement along the orthogonal axis can produce a very good result. However if displacement is done along the normal (or some arbitrary axis) for each vertex it is possible for vertices to overshoot each other, creating a surface with overlapping triangles. If the original surface is not flat, which would be very likely in a tunnel environment, this problem is even harder to avoid. Restricting displacement to some max distance might help to alleviate this problem but this also places restrictions on how far a concrete surface can be extruded from the original wall.

4.2.4 Texture coordinate displacement

Texture displacement is a group of techniques that can greatly improve visual quality. It does not change the geometry or affect vertex positions in any way. There are many different methods for doing this and most work by shifting texture coordinates to achieve a parallax effect. All techniques described below associate a height field with the surface. During rendering a ray is cast from the viewpoint through each pixel. The distance the texture coordinates are shifted depends on where the ray intersects the height field of the surface. Some techniques also enable self occlusion and self shadowing. The benefit of using texture coordinate displacement methods is that the underlying surface complexity can be fairly low and still give the appearance of a complex surface.

Parallax mapping (13) is an approximate technique used to achieve an illusion of parallax and depth. It works by sampling the height map at each pixel and displacing the texture coordinate depending on the angle between the incident view direction and the surface normal. While it is a very fast method for achieving fairly good looking results, this technique cannot achieve occlusion or self shadowing. Because of this, it was decided not to be used in the program.

Enabling occlusion and self shadowing was determined to be a great factor of realism in the simulation. Realizing this, it was decided to take a closer look at sample based texture displacement techniques. Taking a number of samples along the view ray into the height map texture essentially implements discrete ray tracing through the data. Using this, a more accurate parallax effect can be achieved. It is also straight forward to achieve the aforementioned self occlusion and shadowing.

There are many different algorithms for how to compute parallax occlusion. One method described in (14) utilizes a 3D texture to store the closest distance to the height field for any point. When stepping along the view ray this information can be used to determine how long each step should be. This approach is good because it lessens the number of sample steps needed during ray traversal as well as providing a more accurate way of knowing when the surface is hit, effectively making it less prone to aliasing artifacts. On the other hand, a pre-computing pass is required to calculate the distances stored in the texture. For the application of shotcrete rendering where dynamic height fields are used, re-computing a 3D texture with correct distances is simply not efficient enough to be useful.

Another approach is to simply step along the view ray with a predetermined number of steps and checking for intersections with the height field for each step (15) (16). When the height field has been intersected, the parallax distance can be calculated. This requires no pre-computed data but in order to avoid aliasing issues, the number of samples needed is fairly high. Aliasing issues are more likely to occur when the viewing angle is steep in relation to the surface. In (15) this problem is alleviated by adjusting the number of steps dynamically in the pixel shader depending on the viewing angle. To produce antialiasing, this approach also saves both the stepping points around the intersection and interpolates between these two texture coordinates.

This intersection and shifting of texture coordinates are done on a per pixel basis. Because of this, the surfaces can handle fairly frequent changes in height and still produce a very smooth surface which fits concrete rendering nicely. However, if the features in the height field suffer from high frequency changes between adjacent texels, the result can look messy. Because of the discrete sampling of the techniques, a highly varied height map might make the ray miss important features. That is, it can suffer aliasing problems if the number of samples is not high enough to capture the frequency with which the height field changes.

To produce a good result and avoid aliasing artifacts the algorithm needs to take sufficiently many samples while doing the intersection test with the height map. For each sample a texture lookup needs to be made. Because of this, the technique can become computationally heavy.

4.2.5 Our Solution

After performing tests with different kinds of surface rendering techniques it was decided that parallax occlusion mapping would be the best way to visualize the structure of applied shotcrete. The major problems with the other texturing techniques being that simple texturing or normal mapping do not achieve the desired extrusion of the surface. Vertex displacement was also tested but was deemed unfit for the purposes of this program.

For the final concrete rendering in this project, a combination of parallax occlusion mapping and normal mapping is used in accordance to (15). The parallax mapping handles the drawing of the rough features of the concrete while normal mapping is used to give the wall a bit more structure at close distances.

The parallax rendering algorithm starts by transforming the view direction into tangent space and a maximum texture offset distance is calculated. The tangent space vector is normalized and multiplied by the calculated distance to create parallax offset vector. Using the known bump height and the angle between the view vector and the surface normal, the parallax distance can be calculated with standard trigonometric functions. The bump height is an artist controlled parameter which decides the maximum possible extrusion of the surface.

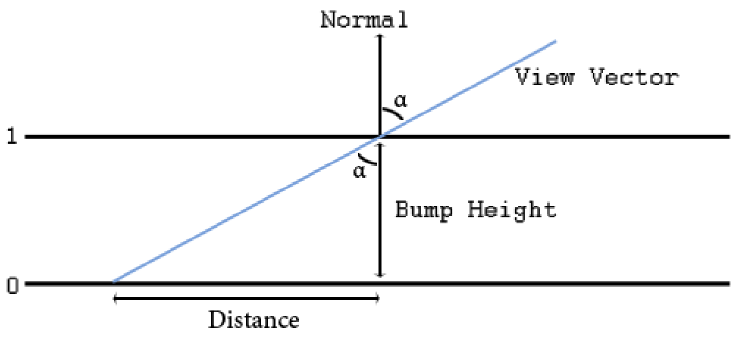


Figure 2: Parallax distance calculation.

Offset calculations of texture coordinates are done in two iterations as described in (17). In the first iteration, a fixed number of steps are taken along the parallax offset vector until an intersection with the height field is found. When this intersection is encountered a second iteration is made with smaller sized steps between the two points around the first intersection. This substantially narrows down the total number of steps needed compared to other parallax occlusion algorithms. The algorithm, as described in (17), uses 16 steps in the first iteration and 12 in the second. We have also found that these numbers produce a good looking result without aliasing artifacts. Compared to the need for 50 or more steps as needed with the naive approach this technique is much more efficient.

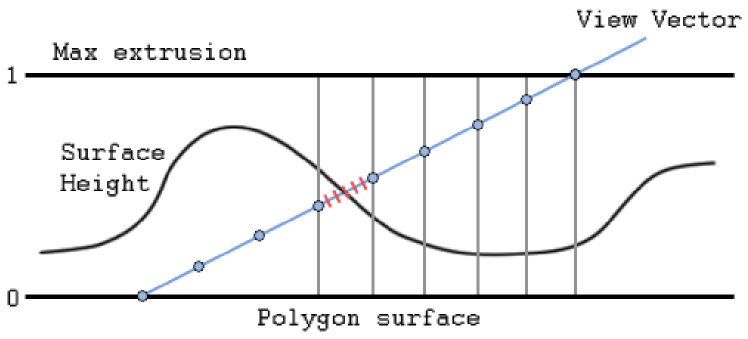


Figure 3: Parallax offset calculation.

As described previously, the updates of the height texture need to be carefully managed. As heights are saved as textures with values from zero to one, extrusion is limited to some max distance. In this case, the problem manifests itself if the maximum height is reached while dynamically updating the texture. This will produce a noticeable seam as the transition from max height to a lower level does not become smooth enough. Parallax mapping have some parameters that can be tweaked, the most prominent of these is the bump height. A small bump height means a small extrusion from the polygon surface, which can reduce aliasing artifacts caused by having a sample based algorithm. A larger bump height will make the altitude alterations more prominent but can lead to severe artifacts if set too high, unless the number of steps is increased accordingly. Therefore, some balance between these two extremes and the amount concrete which is applied must be found.

4.3 Adhesion

As concrete is projected onto a surface, one can distinguish between two major parts; the amount of concrete that sticks to the surface and the amount which rebounds and is wasted. The ratio between the stuck and wasted concrete is called adhesion. Obviously, to make the shotcrete process as efficient and cost-effective as possible, the desire is to keep this ratio as high as possible.

4.3.1 Determining adhesion

The adhesion ratio is dependent on a number of different factors and can, given enough time and processing power, be calculated with precise results. Depending on the level of physical correctness, however, there are different means of calculation one can employ to achieve desirable results. According to (1), some of the factors which govern the adhesion ratio are:

- Geometrical variation of the surface
- Nozzle angle to surface
- Nozzle distance to surface
- Viscosity of the concrete
- Radial angle of spread of the nozzle
- Thickness of the current layer of concrete on the surface
- Velocity of flow of the concrete
- Accelerant dose

Of these, the most important factor is the angle to the surface (18). A perpendicular angle to the surface is preferred since rebound will be minimized and a proper compaction of concrete will be achieved.

Primarily, there exist two types of mixes of concrete used for shotcrete; dry- and wet mixes. When using dry mixing, concrete powder is pumped through the hose and water added immediately before the concrete leaves the nozzle. The water added is controlled by the operator. Wet mix shotcrete uses concrete that have been prepared beforehand with water and concrete mixed together.

The cleanliness of the surface area is also an important factor (19). If a surface has been cleaned before starting the shotcrete process, adhesion will be improved. Conversely, a dirty surface area can greatly decrease adhesion.

The nature of the surface’s coarseness can also be meaningful. Since the contact area between the concrete and surface will rise if the surface is coarse, this will improve adhesion.

The type of rock and mineral composition is a factor as well. Some types of rock have different textures than others and will contribute to determine adhesion of the concrete.

An accelerant compound can be added to the concrete mix. This compound will accelerate the process in which the concrete is hardened, which can have a major role in durability requirements and adhesion. Typically, accelerant is used for shotcrete application on ceilings and overhang areas where the concrete could otherwise fall of if not hardened quickly. Accelerant works fast and is usually added when the concrete leaves the nozzle.

4.3.2 Numerical methods

There has been significant research on how to accurately simulate rock and fluid mechanics in the field of engineering and tunneling. A common way of doing this is by applying numerical analysis (that is, to compute an approximate but hopefully accurate result of a continuous problem by discretizing it). Several different models have been developed for this purpose.

One commonly used method is called the Finite Element Model (FEM) which is a method used to find approximate solutions to partial differential equations. It has been used to compute many rock engineering problems with good results (20).

The Distinct Element Method (DEM) is another commonly used model which works by simulating many deformable (or rigid), particles that interact during the simulation. The DEM method have been applied specifically to shotcrete and yielded good results (2). Here, a particle composition consisting of an outer shell of mortar and an inner coarse aggregate compound is used. The inter-particle interaction is modeled using stiffness springs to accurately model elasticity, viscosity and shear.

Due to the heavy computational power required by these simulations, it is not considered as a viable method in a real-time simulation (1). Because of this, another approximate solution is used in this thesis.

4.3.3 The Adhesion System

Since finding a very accurate solution to the adhesion problem is not viable in a real-time application, other options must be explored. According to (18), the most significant factors in determining adhesion are:

- Nozzle angle to surface
- Nozzle distance to surface
- Accelerant dose
- Area of application in tunnel

This serves the purposes of the application well, since these parameters can easily be found or computed during the update of the tunnel. These are primarily the variables that are of concern in our implementation. Using the results from (18), the adhesion ratio is calculated approximately with a simple function based on the graph in Figure 4. This simple approach was chosen due to its computational simplicity and because it based on empirical analysis of the problem domain and can be considered accurate enough in the current phase of development.

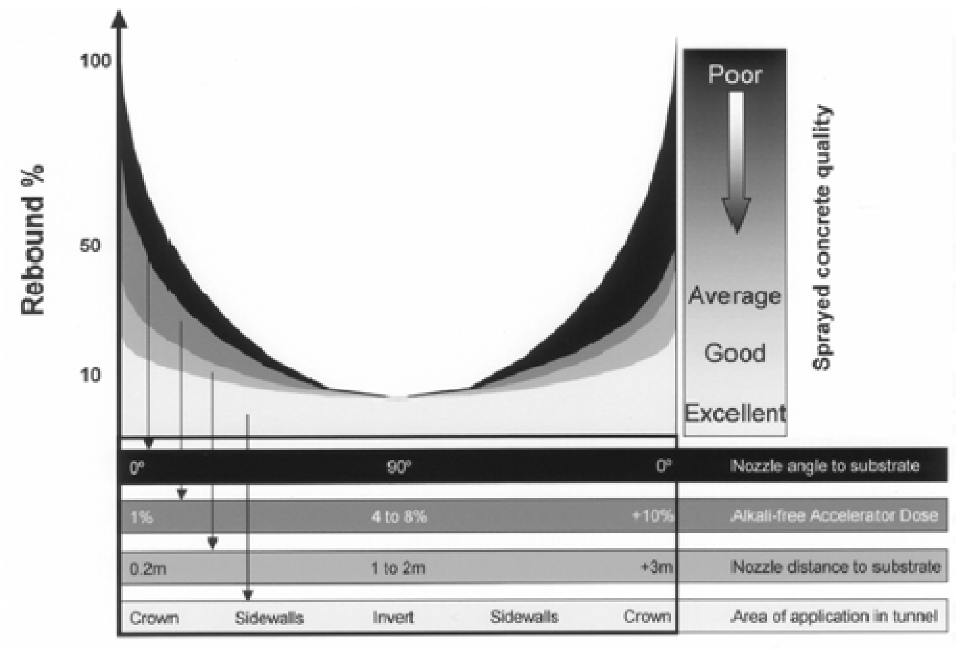


Figure 4: The major factors affecting adhesion. (18)

4.3.4 Design

Since it has been shown that several methods for computing adhesion are possible, with results that are quite distinct in terms of correctness, the design of the system should reflect the different requirements of these methods. While numerical approaches may be infeasible today, the system design should allow the use of such adhesion calculations in the future.

The implemented system defines an interface, AdhesionTest, which is responsible for calculating the adhesion, given a set of variables. First of all, there exist a number of environmental properties which are used in the simulation, see appendix A. These properties govern the general structure of the surface and define characteristics of the concrete used. Properties include geometric variation of the surface, viscosity of the concrete, accelerate dose, and others. Although the simple implementation uses only a handful of these, other variables are included in the system to accommodate future needs. If needed, the data structure that governs these properties can easily be extended to allow for further simulation requirements. As the simulation runs, certain variables are updated and fed to the adhesion test system. These variables are;

- Surface angle
- Distance to the surface
- Thickness of the concrete already placed

Using the environmental and variable properties, the adhesion system has sufficient information to compute the actual adhesion.

Inheriting the AdhesionTest interface, the application implements a simple adhesion test which computes adhesion using basic parameters and approximates the result according to (18).

In order to make the system as general as possible, and make future implementations easy, a factory class is used to create the actual adhesion test. Future tests need only extend the AdhesionTest base class, and add a few lines of code into the factory class. The choice of the system can then be made via a setting read from a file, for example. An overview of the adhesion system can be seen in Appendix A.

4.4 Tunnel Design

The rendering and adhesion systems have introduced a number of requirements for how the actual tunnel must be designed. In order to create a functional system, these aspects need to be taken into consideration. This section describes how concrete data is stored and how this data is updated.

4.4.1 Tunnel Geometry

For the simulation to be realistic, the environment in which an operator works need to be taken into consideration and be depicted by an accurate graphical representation. Tunnel geometry must somehow be obtained and rendered.

There exist 3D models which have been constructed by laser measurements from real world environments. This is perhaps the most accurate way of representing a tunnel, but it is not without drawbacks. A finely tessellated geometry mesh can in the worst case negatively affect the performance of the program. This is unlikely to be a problem in this program due to the relative low amount of geometry needed. Also, the OpenSceneGraph library provides tools for simplification of model meshes which can potentially be used to great effect. However, these features were not used in this implementation.

A second solution is to procedurally generate the tunnel mesh, which was the method used in the project. The advantages of this technique are, in the case of tunnels, numerous. This way, the tessellation of the mesh is easily adjustable which can be a contributing for doing tests of different kinds. Because of the general homogeneity of the structure of a tunnel, generating a tunnel mesh procedurally is fairly straight forward. Many tunnels can, if simplified to an extent, be viewed as a half cylinder with a floor running between the two halves. This gives us the ability to use a second order equation to generate the basic shape of the tunnel. An offset parameter is then used to displace the vertices slightly to give the tunnel a more irregular and natural look.

4.4.2 Storing Concrete Data

A fundamental requirement for the program is the ability to store concrete information in some way. When concrete gets shot at a surface, it needs to be stored and later updated, rendered and sampled for statistical purposes. The most intuitive way of storing this information was to use textures to store concrete data in the form of height maps. This means that concrete elevation information is represented by a 2D, grayscale image.

Height maps are represented as regular textures, and as such, they are imposed the same problems and restrictions. A texture is composed of a finite number of texels, depending on its resolution. When applying a texture to a surface, an individual texel will cover a certain area of the geometry to which it has been mapped. If the size of the geometry is increased, the texture needs to be stretched, meaning that one texel will cover a larger area on the geometry.

To cover an entire tunnel with a single texture, the texture needs to have a very large resolution in order to produce sufficiently good visual quality. Otherwise, it will be too stretched to provide enough detail on every point of the surface. This approach is cumbersome and inefficient. A better solution would be to let a single texture cover a reasonably small part of the tunnel. Since concrete data need to be unique for each part of the tunnel, it means that the concrete data texture cannot be repeated over the tunnel geometry as is possible with regular texturing mapping. Instead, many textures are needed to store concrete data, each covering a smaller part of the tunnel geometry.

This leads to a natural segmentation of the tunnel. Each segment handles a part of the tunnel geometry and references its own concrete data texture that covers all geometry of that segment. Rendering an entire tunnel with many concrete textures covering small parts would not be possible due to a limited number of texture units available on graphics hardware. Splitting the geometry into sections takes care of this problem while making things easier to handle and make it more efficient. First, it becomes faster to update concrete data as smaller parts of the tunnel can be updated at a time. Secondly, separating the tunnel into segments makes it easier to render the tunnel as culling becomes more efficient.

The height information is based on a gray scale image. This limits the height differences in the range from black to white, or zero to one. This means that no infinite amount of concrete can be stored in one place, which needs to be taken into consideration. The scale of each update in relation to the height field needs to be carefully balanced because if the height field fills up too quickly, the user experience will be disrupted since no more concrete can be applied.

4.4.3 Updating Concrete Data

It was decided to decouple updates of the height map and the visualization of the shotcrete ray. This implies that the ray of concrete that is sprayed on the wall is not actually responsible for updates that occur in the concrete data. This separation seemed like the most reasonable approach which allows for a good design and a satisfying visual experience. Letting the concrete particles have an actual impact and affect on the surface would put significantly different demands on the system. It would be necessary to somehow store the individual particles that were shot, as well as making them look like a coherent mass of concrete when placed on the wall. This would not be efficient and it is not strictly necessary for the simulation to look visually pleasing.

When concrete is shot, a ray is cast from the nozzle into the world. This ray is intersected with the tunnel geometry and, if an intersection takes place, cartesian information (such as position, distance and surface angle) is fed to the adhesion calculation system which computes the percentage of concrete that sticks to the surface. The concrete is updated in a radius around the point of intersection, depending on the spread of the ray as its distance to the surface is changed. The information can also be used to fetch the tunnel segment which is to be updated. Using this information and the concrete adhesion ratio, the corresponding height map texture can be updated.

As was discovered, colliding only a single ray against the tunnel geometry has a serious drawback. When updating the height data in a radius around the intersection point, this does not accurately represent the physical events. This solution assumes that all triangles in the radius of the beam lie in the same plane as the triangle which was intersected. This might definitely not be the case since the tunnel surface is bound to have imperfections and a generally uneven surface. To get a more distributed application of concrete, a solution with several adjacent beams were implemented. Several beams with a smaller individual radius are shot in a cone shape from the nozzle of the robot. This improves the probability that adjacent areas are updated using the correct data.

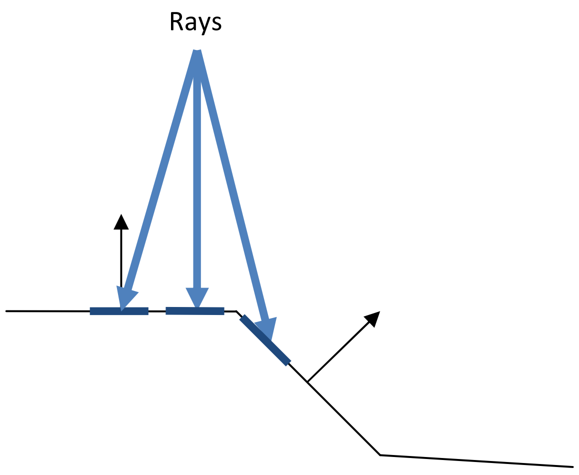


Figure 5: Multiple rays.

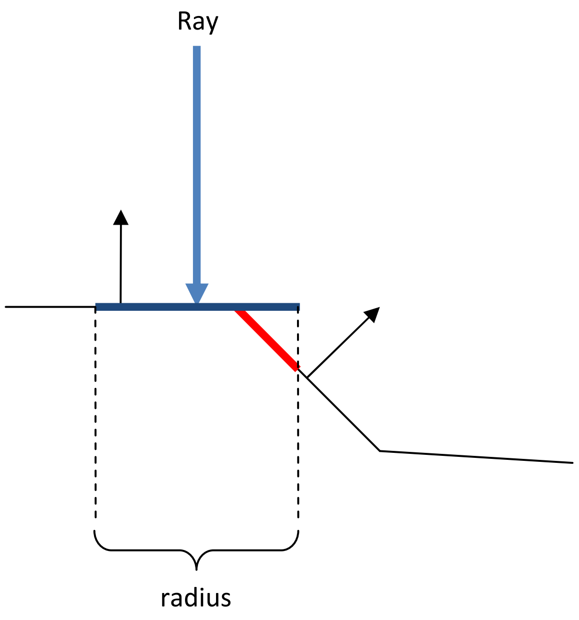


Figure 6: Single ray.

Observations between these two methods were conducted. Comparisons were made in terms of visual artifacts and it was noted that single ray method was only marginally less attractive than the multiple ray method, but at a significantly reduced complexity. Therefore, a decision was made to stick with only one ray.

Since the geometry of the tunnel is divided into several segments, there are some minor problems in updating a height map if the point of update is too close to an edge of another segment. At first, this problem was partially solved as a side effect from the above by shooting several rays in close proximity to a central ray, in an area around it. This, however, was not at all an optimal solution. It did ease the problem of updating neighboring segments, but it did not solve it. Furthermore, this solution proved to be difficult when determining exactly how much concrete that was applied to a certain area since the surrounding rays would influence the central one in unexpected ways. A far better solution was implemented as a consequence of the single ray solution from above. This consists of simply checking the bounds of a segment and updating the texels of concrete textures in neighboring segments if required.

4.4.4 Performance Considerations

Updating the height maps can be a major bottleneck in the application if not handled carefully as it is expensive to send data to the graphics card. The size of the texture which is updated is the most important factor which contributes most to a difference in performance. The texture has two channels, one for concrete height data and one is used for storing wetness information (see section 4.5.4). The color information of a pixel is stored in 4 bytes, so a texture resolution of 512^2 pixels means a memory requirement of 2*4*512*512 ≈ 2MB. With a resolution of 1024^2 the memory requirement is four times higher. Setting the height texture resolution too low can cause artifacts to appear when rendering as individual texels can be seen. It is therefore important to balance the visual quality versus the performance. It was found that using a resolution of at least 512^2 pixels is required to provide a visually pleasing experience. A resolution of 256^2 pixels can break the illusion of concrete as it is not high enough to create a smooth enough concrete surface. A resolution of 1024^2 is, however, a bit of a stretch seeing as the added visual contribution is hardly noticeable, but the performance loss is.

All these texture resolution tests are dependent on how large a tunnel segment is in actual geometry. If a segment size is doubled, the resolution of the concrete texture would also need to be doubled to still provide the same visual quality. Therefore it is important to find a balance with tunnel segments that have a suitable size and textures with a resolution that produces a good result while maintaining visual quality. Hardware filtering of textures ensures that the concrete data is always rendered correctly with good visual quality at any distance.

4.5 Shading Model

This section describes various shading techniques used throughout the simulator. The texture creation process and evaluation is also deliberated. Performance consideration of shaders and algorithms are also considered and examined.

4.5.1 Overall model

Geometry in this simulation exists in two basic forms, the tunnel geometry and the shotcrete robot. The lighting model used in all shaders is the Phong lighting model (21). As the focus of the work has been to visualize concrete, the tunnel geometry uses a few other more advanced effects as well.

When computing the color of a pixel in the shader used for the tunnel geometry, the parallax effect is only computed for pixels within a certain distance from the camera. This saves a significant amount of computational power since most of the pixels on the screen will often be far from the viewer so that the parallax effect is not prominent enough to warrant a full run through the parallax occlusion code. Instead, the original texture coordinates is used, computing only shadows on the surface. As tests showed, still computing shadows on the surface is necessary to avoid sharp edges where shadows suddenly would pop into existence.

The simulation uses two height fields during rendering. One is used for the stone surface of the tunnel which is repeated along with the diffuse and normal map texture. The other is used for storing the applied concrete. Using separate height fields is necessary to be able to use parallax occlusion mapping on both types of surfaces while still being able to keep track of the amount of applied concrete. While no concrete is applied it is trivial to use the stone height field while rendering the surface. On the other hand, when concrete is applied, it is not possible to only use the concrete height field for parallax computations. This would produce an undesirable visual appearance since it would look like the concrete was applied directly to the polygonal surface, disregarding the stone surface. Still, both surfaces need to benefit from parallax mapping to produce a realistic environment. So, while rendering concrete surfaces, the height from both the stone and the concrete height field is used. This effectively doubles the amount of samples the shader needs to take during parallax computation of concrete covered surfaces. Given sufficiently capable graphics hardware, this is not a problem but it needs to be considered should the system be applied to a low-end machine.

After correct texture coordinates have been calculated, the correct texture lookups are performed in order obtain a color value and normal for both the stone and concrete materials. The parallax occlusion mapping takes care of modeling the large features of the materials and the normal sampled from texture is used in the lighting computation to simulate small features in the materials. With the correct texture coordinates soft shadows can be computed and with all this information full lighting computation can be done according to the Phong shading model for all light sources affecting the pixel.

4.5.2 Texturing

Textures are essential components in making surfaces look realistic and are a basic way of visualizing materials in computer graphics. This section details the creation and application of textures in the simulator.

4.5.2.1 Concrete

The environment in the simulator oftentimes makes the user view a large portion of the tunnel surface. This puts a certain requirement on the textures of the tunnel in that they need to tile well. That is, the textures should not exhibit edges or other distinct visual artifacts when put side by side. In a virtual environment, a proper tiling of textures is often required but it is particularly important in the case of this simulation.

When constructing a concrete texture, visual references were used as a base in order to achieve a natural look. Actual photos of concrete were tried and tested only to be discarded in many cases. Most times, these textures displayed many artifacts which made them unfit for use. Such artifacts often manifested as distinct visual attributes which made the texture look very artificial when tiled. Also, photographs often featured dust, scratches, lighting and dents in high frequencies which did not make for a good look when put on a virtual surface.

Instead, attention was turned to the creation of concrete textures using a procedural approach. Concrete has a quite even color though with slight variations. To model this, a cloud-like texture shifting between two similar concrete-like colors produce a nice diffuse color for the concrete. This approach tiles better and models reality in a better way. Producing a cloud-like image can be done with fractal Brownian motion (fBm), in this case based on Perlin noise (22).

Figure 7: Perlin noise based fBm in black and white.

In the end, the textures that are used were created with image software as it was simpler to tweak different parameters and making the texture tile well. Something that was discovered was that the concrete texture only required subtle shifts in color to look good. If the contrast between color shades were too great, the result would be that the surface looked shadowed which is not desirable. Since shadows are important for the human sense of understanding spatial relationships (23), this was only confusing.

To achieve a proper coarseness of the concrete surface, a normal map was applied. Again, tests were made in order to achieve good look. Different levels of bumps were tested and compared to actual photos. In the end, the map that was used was created as a combination of image references and what was aesthetically pleasing.

Figure 8: Concrete normal map.

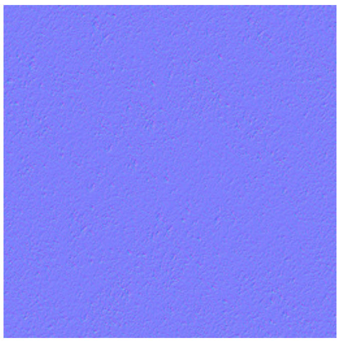


Figure 9: Concrete diffuse texture.

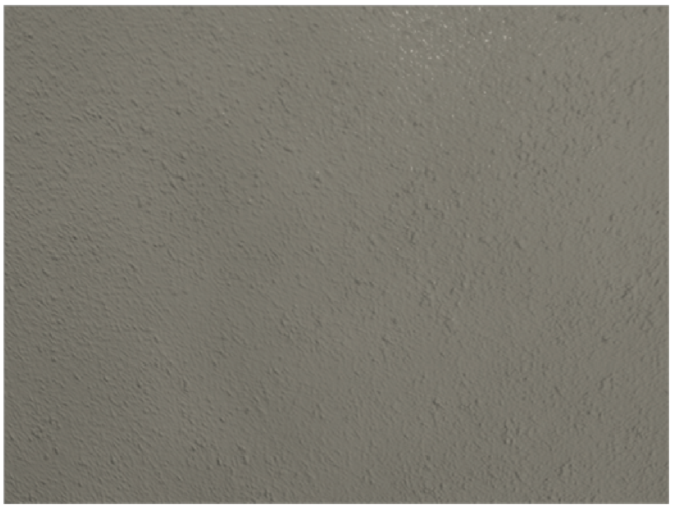


Figure 10: Rendered concrete.

4.5.2.2 Stone

A procedural approach was also employed when creating textures for the stone surfaces. This, however, was not as successful as for previous textures. The purely procedural textures that were created looked very artificial and, while resembling stone, did not have the desired visual quality. Instead, photographic textures were used to great effect. In order for the stone to tile well, the photos were manually retouched. The photo that was used as the starting point was that of gneiss which is a common type of rock.

The usage of parallax mapping on surfaces requires that a height map be created. This was a process that underwent several iterations and tests. To get a pleasing result, it was discovered that high frequency changes in height made the stone surface look very edgy and sharp. To rectify this, more subtle shifts in height were added and a slight blur effect was added to tone down frequent changes. The height map was also modified to simulate drill holes in the stone.

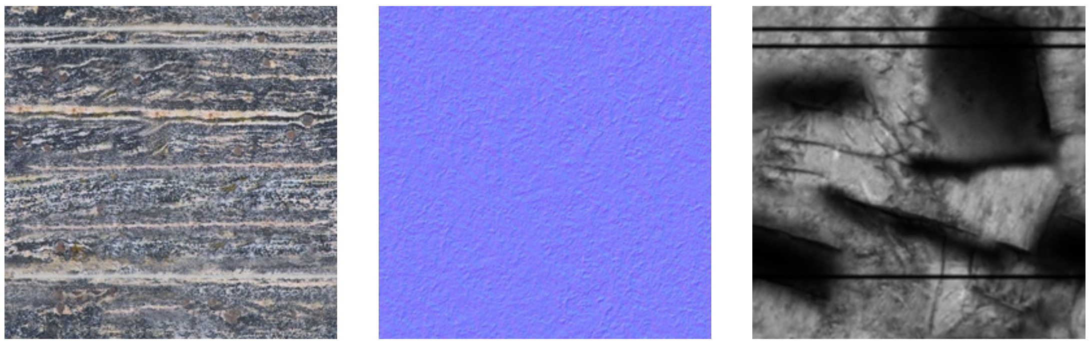


Figure 11: Stone components. From left to right: diffuse, normal and height map.

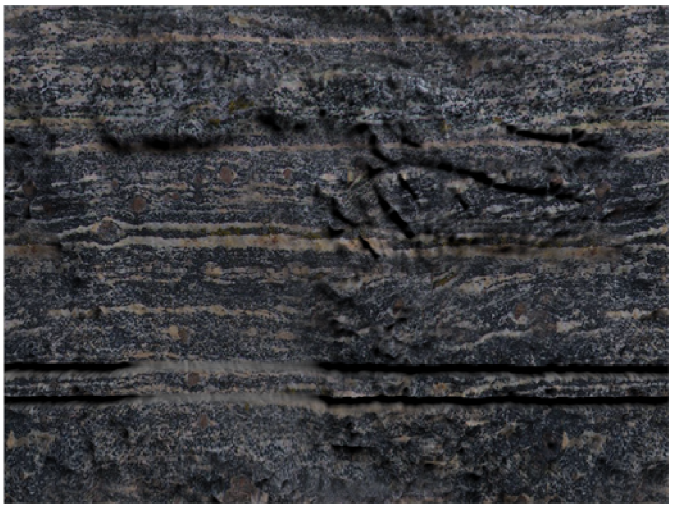


Figure 12: Rendered stone.

4.5.3 Soft shadows

Another benefit of using parallax occlusion mapping is that the height fields used during those computations can also be used to produce good looking soft shadows. This further improves on the realistic feel from parallax mapping. Without the ability for the height field to cast shadows, the parallax effect might even look at bit strange as one expect the different height in a object to cast shadows.

In the real world, the abundance of soft shadows that exist around us is a result of having light sources that cover an area. Infinitely small light sources are often used in simulations to keep things simple, but to be physically correct it is impossible for such light sources to produce soft shadows. The method used to calculate soft shadows for the height fields is implemented as described in (15) and it is an approximation of the real shadows but it does produce a believable result. Shadow in one pixel is done much in the same way as the parallax occlusion computation. First, a vector is created from the pixel towards the light source. This ray vector is then traced from the surface toward the light source and checked for intersection with the height field. In the next step each sample height is compared to the height of the pixel and an approximate shadow value is computed from the sample with the highest amount of occlusion.

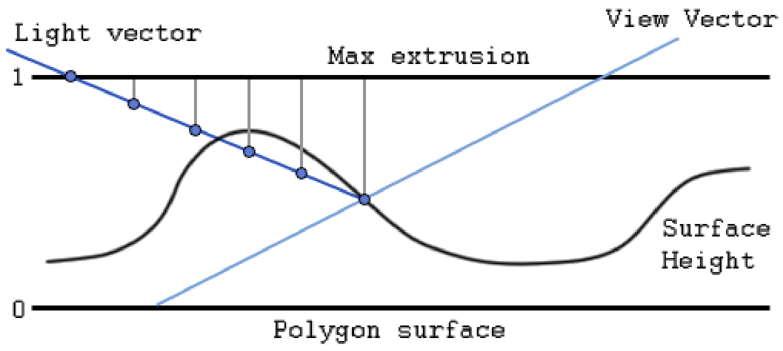


Figure 13: Soft shadow calculation.

Just as the parallax effect, the shadows are also computed using the combined height field from stone and concrete. When the concrete height reaches a certain threshold, the shadow computations switch from using the combined height field into only using the height field from the concrete. This is done to give the surfaces covered with concrete a smoother look. With this computation, the roughness of the stone height field will not contribute to the shadows seen on a concrete surface which would otherwise be the result. This in turn will make the concrete appear to fill out holes in the stone surface and produce a smooth concrete surface.

4.5.4 Wet Concrete

The concrete mix that is projected from the hose nozzle contains a mixture of concrete powder and water. This is necessary in order for it to stick to the target surface. As the concrete hits the surface, it should gradually dry and become solid. This effect is simulated in the program as follows.

The texture map which stores height information for the concrete information only requires one channel of color information. This leaves an additional three channels unused, a fact that we exploit and in one of these channels we store a time value. Each time the texture is updated with concrete information, a timestamp value is also stored. The texture is passed to the fragment shader which looks up the information and uses it to shade the pixel in an appropriate color, enabling it to look wet.

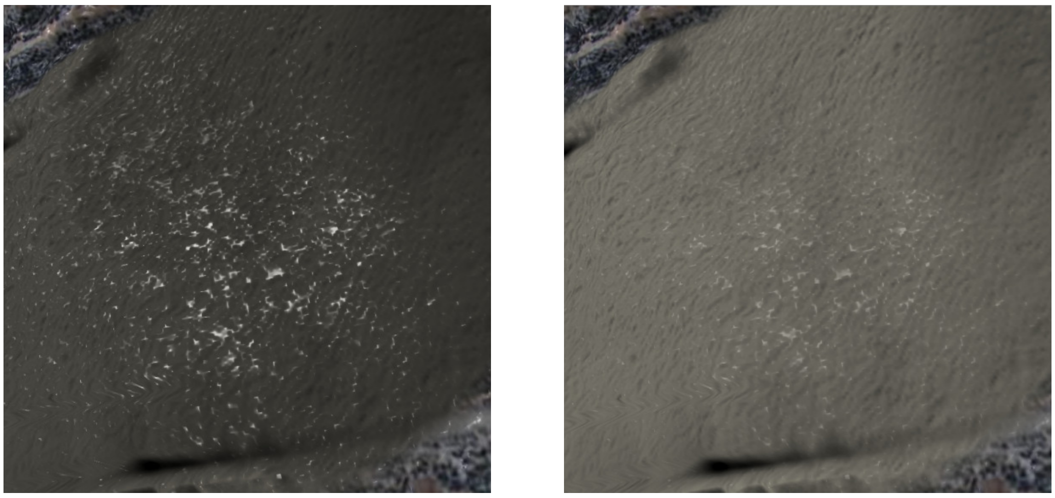


Figure 14: Left, wet concrete. Right, dried concrete.

The actual shading process is very simple and no actual physical simulation is done to produce a correct result. If the concrete is wet, the diffuse color is darkened and the specular highlights are accentuated. As the concrete dries, the effect fades away.

4.6 Particle Systems

In this project, the shotcrete ray and its associated effects are the kind of objects which can be hard to render as standard geometry. Instead, particle systems fit very well to simulate these effects (5). The rendering engine used, OSG, supplies a basic implementation of particle systems. However, this system only supports square billboard particles, something which were deemed unfit for the purposes of this simulation. Therefore, a new implementation of particle systems was done.

The exact effects that are rendered as particle system are the following:

- Concrete shot from the nozzle of the robot.
- Mist, or dust, that is produced as a result of high pressure impact on the surface.
- Concrete particles that rebound when striking a surface.

Each of these effects is implemented as a separate particle system. They are then combined to create a visually pleasant experience.

4.6.1 Implementation details

The particle system implementation for this project is fairly straight forward. Advanced effects, such as interactions with the tunnel geometry, are not implemented since they were deemed unnecessary. An overview of the particle system can be seen in Appendix A. The particle system class manages all living particles and keeps an emitter, which as the name suggests, emits particles into the system when required. Each emitter is assigned a template particle which is used whenever a new particle needs to be created. The template determines the properties and visual appearance of a particle.

The particle class can handle two types of particles; Camera aligned billboards and directional aligned. Direction aligned particles will rotate and align along their velocity vector, as opposed to a billboards rotation that always relates to the camera. The main reason to do this was so that it would be possible to animate the texture coordinates of particles in the shotcrete ray along the velocity vector. This becomes trivial if the particles are already aligned along the velocity vector. The animation will then only consist of moving the Y texture coordinate according to time.

To be able to simulate particles that are affected by forces, such as gravity, each particle system can have a force associated with it. This force is simply applied to the velocity of each particle alive in the system during each update.

4.6.2 Texturing

A real-life shotcrete ray consists of a large amount of small concrete particles. This is handled by the use of particle systems in itself, but it is not enough to achieve a pleasant look. In a first try, a texture with a single fuzzy cloud image were used, but it was found that this did not achieve the desired, somewhat chaotic, look of the large amount of particles that needed to be simulated. One approach to alleviate this problem is to let one particle in the program simulate many real world shotcrete particles, similar to what is used in (17). Here, each simulated particle is textured with an image representing many small particles. This helps to maintain the illusion of a great number of particles. At the same time, it keeps the amount of geometry rendered to a minimum, something that can be important for performance reasons.

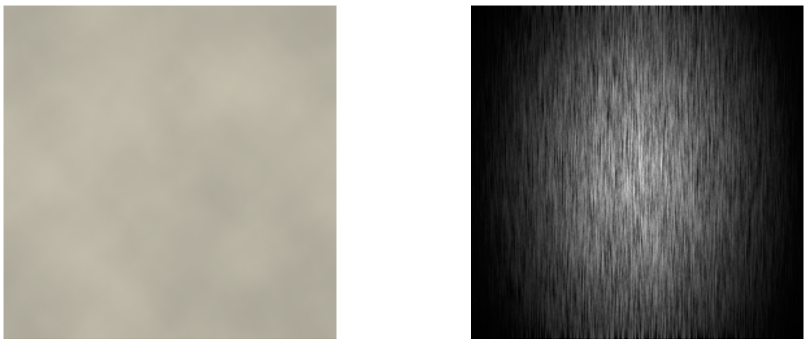


Figure 15: Ray particle components. Diffuse texture and alpha channel.

If the particles in the system are moving very fast it can become troublesome during the update of the system. Particles will move along their velocity vector during each update which can create visible gaps between chunks of particles, especially if all particles have similar velocity. To avoid this, the velocity of the particles needs to be kept down at a reasonable rate. However, it is still desirable for the particles to be perceived as moving at great velocities like their real life counterparts. To improve the sense of speed, the texture is animated along the velocity vector of the particle as described above. As stated, each particle is texture with an image consisting of many small particles. Yet another trick to enhance the perception of speed is to blur the texture in such way so that it appears to be in motion. The result of these tricks is that particles can move at a low enough speed so that the shotcrete ray will stay continuous but still convey a sense of high velocity. The combination of these effects will produce a realistically looking shotcrete ray, seemingly consisting of a great amount of particles moving at very high speed without producing immersion-breaking visual artifacts.

4.6.3 Soft Particles

Particle systems are used to simplify implementation of graphical objects in order to save performance. Most particles are often implemented as billboards, which simulate 3D volumes by projecting them to a 2D plane. Billboards can often be used with great results. However, the illusion of volume can easily break when the simple 2D image is intersected with some geometry, giving birth to ugly artifacts manifested by sharp edges. This effect can be diminished by using a larger number of smaller particles. However, this is computationally more expensive.

In order to remedy this, the sharp corner that the intersection creates can be smoothed out, or made soft so that the illusion of volume is maintained. One way of achieving this is by blending the particle depending on the difference in depth between the particle itself and the world geometry (24). This is done per pixel.

In the simulation environment, we use this soft approach to render all particle systems. Smoke and dust particles, which are rendered as big, slowly moving billboards, are significantly improved visually by using this technique. The shotcrete ray, which often collide with the rock surface it is aimed at, also benefit from this. Since the ray is composed of fast moving, small particles, the effect is not as easily noticeable.

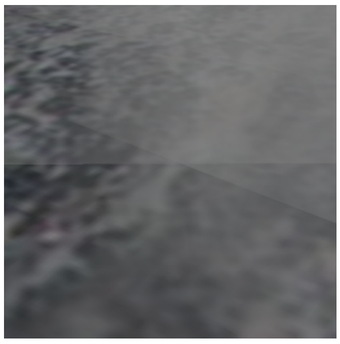


Figure 16: Hard particles intersection geometry

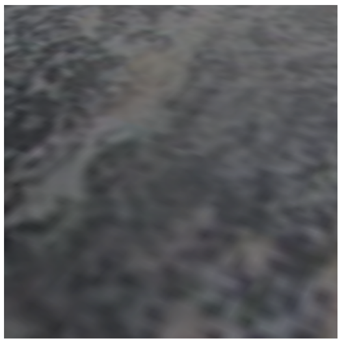


Figure 17: Soft particles intersecting geometry

4.6.3.1 Retrieving and Using Depth Information

In order to retrieve depth information, an auxiliary render target is used. The actual rendering consist of two passes. The first pass draws the scene normally but without rendering particle systems. In the shader stages, the (view space) depth of the world geometry is calculated and stored in the auxiliary buffer. In a second pass, all particle systems are rendered. The pixel shader of these particles sample the auxiliary buffer previously populated by depth information and calculates the appropriate blending by comparing the depth in the buffer with its own depth value.

The downside of this approach is that all objects that make up the world geometry must have their shaders modified to write the view space depth to a render target. The bright side is that the auxiliary depth buffer can be sampled by other shaders when creating other types of effects, such as post processing effects.

Another method that can be used utilizes the native depth buffer. This method requires no extra render targets but instead recreates world (or view) space depth in the pixel shader of the particle systems. The downside is that for each pixel of each particle, depth information must be reconstructed, which can degrade performance when displaying many particles.

Tests that were made did also show that the standard 8 bit depth buffer is not precise enough and particles suffer from heavy artifacts when precision is decreased. Therefore, 16 bit depth is used, which in this case works fine. It is possible that a 32bit buffer would achieve even better results but our tests showed that this did not produce any noticeable difference in results. It should be noted however, that this application is not dependent on a high view range. This should be taken into consideration when choosing the precision of the depth buffer as well as when setting the near and far clip planes.

4.6.3.2 Blending the billboard

Since depth testing is necessarily off when rendering alpha blended particles, all pixels of all particles will have their fragment shaders run. Obviously, the number of instructions in these shaders can significantly impact the performance of the program. Retrieving or calculating the depth value for a pixel can be more or less advanced, but another bottleneck can be the way which the pixel is blended with the background.

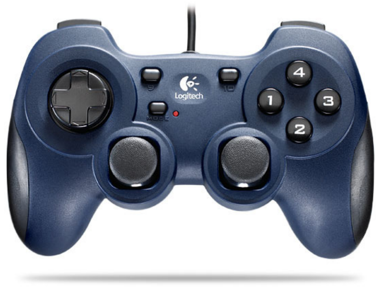
Comparisons between different techniques for blending were made with varying results on visual quality and performance. Three different kinds of approaches for blending are suggested in (25). The first one uses a simple and cheap scaled difference, which the authors say can lead to artifacts in some situations. The second method uses a contrast function (as they call it) to achieve a smoother fade and reduce artifacts. The third method uses another “smoothing curve” to reduce artifacts. This function, however, can lead to some artifacts as well, the authors predict.

For the purposes of this simulation environment, our experiments show that the first function of (25) is to be preferred (See equation below). First, it has a lower computational requirement than the other methods. Secondly, it does not suffer from conditionally executed code, or branching. Branching could be a significant cause of lost performance on the GPU since it is inherently parallel (26). Indeed, our experiments showed a small loss in frame rate when the second and third options were used but virtually no difference in visual quality. Therefore, the choice was made to stick with the first, cheapest, option. This is also the method used in (24).

4.7 Input

Controlling a shotcrete robot is most often done through a remote control with joysticks. Even though most developers of shotcrete robots have different control setups, the principle of these are very similar (1). The controls are in most cases connected through a Programmable Logic Controller (PLC) which is a reprogrammable interface that controls input and output data. The PLC can also be connected to a standard personal computer which indeed makes it possible that this kind of device can be used in a simulated environment.

For a fully functional simulator it is of course necessary to be able to use a real remote control but it is not strictly necessary during this stage of development. During this early development phase, no effort was made to find and employ an actual robot controller. Instead, a simple game pad was used as a substitute. The game pad in question was the Logitech Dual Action game pad (27), which has dual analogue controls that provide the same degrees of freedom as real shotcrete robot remote controls.



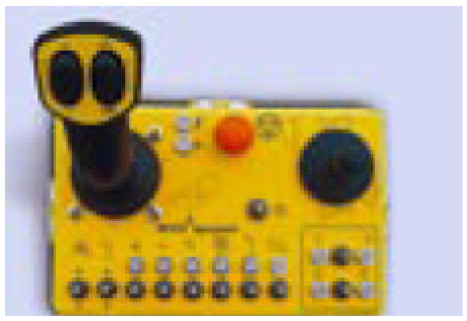


Figure 18: Left, Logitach Dual Action (Source: Logitech). Right, Meyco Potenza control device (Source: Meyco).

To capture input to the simulator the Object Oriented Input System (OIS) was used (28). This is a free library for platform independent input handling. In this case it uses DirectInput (29), as the simulation program runs on Windows where OIS is wrapped around the DirectX input system.

In order to decouple external input libraries as much as possible, the simulator manages its own internal input messages and structure. The OIS messages are translated to the internal format and, at a slight performance overhead, generality is kept high. Handling input this way is important since it allows for a smooth transition to a new system in the case that this should be necessary. For example, in the future it is likely that a proper control device will be required, in which case a switch to some PLC interface would be necessary.

The robot is controlled using the two analog joysticks and some of the keys on the game pad. Preferably, the control scheme should be intuitive but at the same time provide the level of control that is necessary. Controlling both the robot and the avatar of the operator at the same time presents a problem with the control input. Since there are not enough keys on the game pad to control both of them, this is done in separate modes. Using a key on the game pad, the operator can swap between controlling the avatar and controlling the robot. While this setup solves the problem in a fairly intuitive way, the drawback is that the avatar and the robot cannot move at the same time.

4.7.1 Robot Design

When looking at shotcrete robots, there are basically two different types of equipment. One type of robot is equipped with a cockpit in which the operator sits comfortably and controls the device. The other version is operated from the outside, with a remote control. The simulation is focused on robots which are remotely controlled. Then again, adding support for static driver positions would be a fairly simple addition.

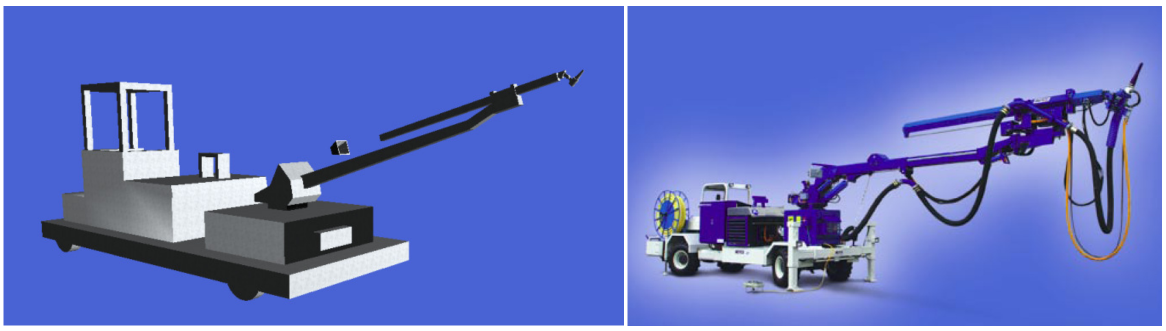


Figure 19: Left, Robot model. Right, Meyco Potenza.

To achieve a simulation as realistic as possible, the ideal situation would be to acquire a real model of a shotcrete robot. Since getting a real robot into the simulation was not high priority, this did not happen during this project. Instead, an approximated model was constructed and loaded into the simulator. The model used was, although being a very simple prototype, constructed to be similar to a real world version. The modeled version features roughly the same number of joints and a similar structure as its counterpart. Because of available material, the robot used in the simulation is modeled after the Meyco Potenza (30).

4.8 Quality Assessment

An essential aspect of a training simulation is the ability to get feedback from the system in various ways. One important feature of the system is to be able to grade how well a shotcrete session was performed by the trainee. What was the result of the task? What was performed well? What could be done better? If something did go wrong or were badly performed, what was it, how can it be improved and what should the trainee do to fix it? After a simulation session has been run, the system needs to be able to answer questions like these.

The system can provide feedback to answer the questions above. This can either be shown after a session or during the simulation to continuously update the trainee on how well she is doing. Another form of help the system can provide is to automate or provide tools to help the trainee perform certain tasks. These tools can be used in the beginning of training and be gradually removed as students become more proficient at the task at hand.

The mark of skilled shotcrete robot driver is the ability to apply the right amount of concrete in an even layer over a surface without rebound and waste becoming too great. Because of this, it is very important that the shotcrete simulation has the ability to accurately report results of how well the driver performed so that it is known how much concrete was wasted and how thick and evenly distributed the concrete is. This simulation provides tools both to help the trainee during simulations as well as the ability to thoroughly inspect the result of the simulation after a session is completed.

4.8.1 Statistics

During the simulation, various forms of statistics are gathered. Information stored consists of the amount of concrete used, how much of it that sticks to surfaces and how much of it that goes to waste. Users are able to show a table displaying these values at any time during the simulation. This makes them useful both during the simulation, as an aid to see how well a trainee is doing, as well as after a training pass to see the total amounts that were used and wasted.

4.8.2 Training Tools

The statistics mentioned above collects values over the entire training session. However, it could also be interesting to find out how well a trainee is doing over a short period of time, in order to get immediate feedback. Users have the option to enable a graph that continuously displays the percentage of used concrete that stuck to tunnel surfaces.

Another training aid that is available is the addition of a laser sight that sits on the nozzle of the robot arm. This helping hand is featured to help the operator point the nozzle in the right direction. To make tests and education more realistic, this feature can be turned off.

4.8.3 Inspection Tools

Not only is it important to know the adhesion ratio during the shotcrete process, but it is also important to know the thickness of the concrete that has been placed on the wall surface. Currently there are two ways of displaying concrete information, with color coded surfaces and with numerical displays.

4.8.3.1 Color Coding

The first option is to color tunnel surfaces in various colors depending on how much concrete has been applied to them. This can be used to display different pieces of information. Currently there are two different display modes for coloring surfaces. The first compares the current concrete amount against a desired depth value and colors the surfaces according to the difference to the nominal value. Three base colors are used and shades between these indicate the current depth relationship. The color scale starts at red, indicating that the thickness is below the nominal value. It then fades to green, which indicates a desired thickness. A blue color indicates that the concrete thickness is too high.

The second color scheme uses the average height of concrete in the whole tunnel and displays how much the current height varies from that average value. The colors fade from green when the thickness is equal to the average and, as before, showing red and blue indicating less and greater thickness, respectively.

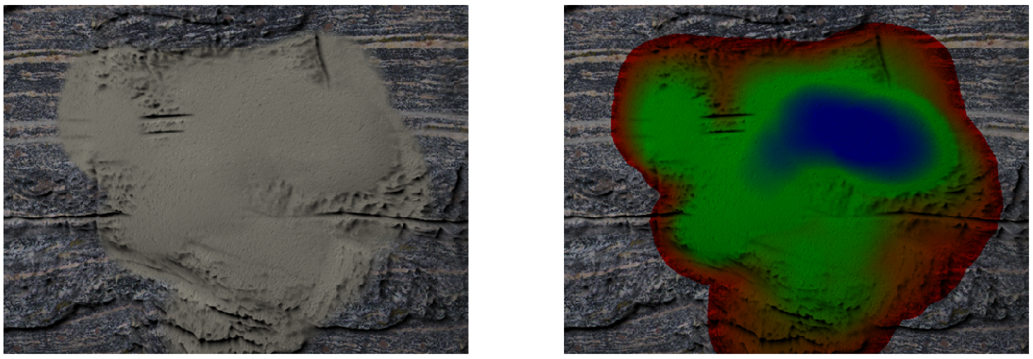


Figure 20: Concrete rendered normally and with color encoded height.

4.8.3.2 Numerical Display

Another way of displaying information about concrete thickness is by placing labels with numerical information on the tunnel surfaces. Instead of taking a sample at exactly one point on the surface, a display label gather statistics in a radius surrounding it. The data sampled can be used in different ways. There are currently two display modes supported in the simulation. One mode displays the surrounding local average of a point. The other mode gathers the average thickness globally and displays the difference in depth compared to the local average.

4.9 System Design

The rendering framework that was chosen to work with is OpenSceneGraph which, as the name suggests, is a structured scene graph. That is, the 3D scene consists of a tree structure with nodes that each is responsible for rendering and transforming different objects. For example, the robot consists of a base node and several child nodes that make up parts such as the arm and the body.

Using a scene graph has certain advantages that were exploited when structuring the rendering system for the simulator. Particle systems need to be drawn lastly in order for them to be displayed correctly. Conveniently, two separate node trees were constructed for this purpose. The first node tree stores all opaque geometry, such as the tunnel and the robot that should be drawn first. The second node is responsible for particle systems only. During the render loop, these nodes get drawn in order which separates the rendering in an easy way. It is also trivial to add or remove geometry in either of these nodes.

In order to manage objects that need to be initialized, updated and destroyed, there exists an interface called Entity. An entity is simply an object that can be updated and rendered. Objects which need this kind of functionality extend this interface and are maintained and cared for by a manager. As a standard way of managing updatable objects, this is a clean and easy approach. Objects that serve as Entities in the simulation include among others, the robot, tunnel, and shotcrete ray.

At the center of the program, a class called World binds all the parts together, which means that it handles the initialization, updating and shutdown of the simulation. It also maintains references to input handlers, statistical tools and the OSG render system, among other things.

5 Results and Discussion

This chapter details the result and problems of various aspects of the project. Comparisons are made between parts of the simulator and their real world counterparts. Performance issues and hardware requirements are also discussed.

5.1 Rendering

During rendering, stone and concrete materials use parallax occlusion mapping. The number of steps used in the parallax occlusion algorithm determines the visual quality of the effect. At greater distances from the camera, fewer steps are needed to produce a good looking result but at close distances the visual quality is degraded if not enough steps are used during calculations. The following images display what a stone surface looks like at a close distance with different number of steps in the first and second iterations of the parallax algorithm, respectively.

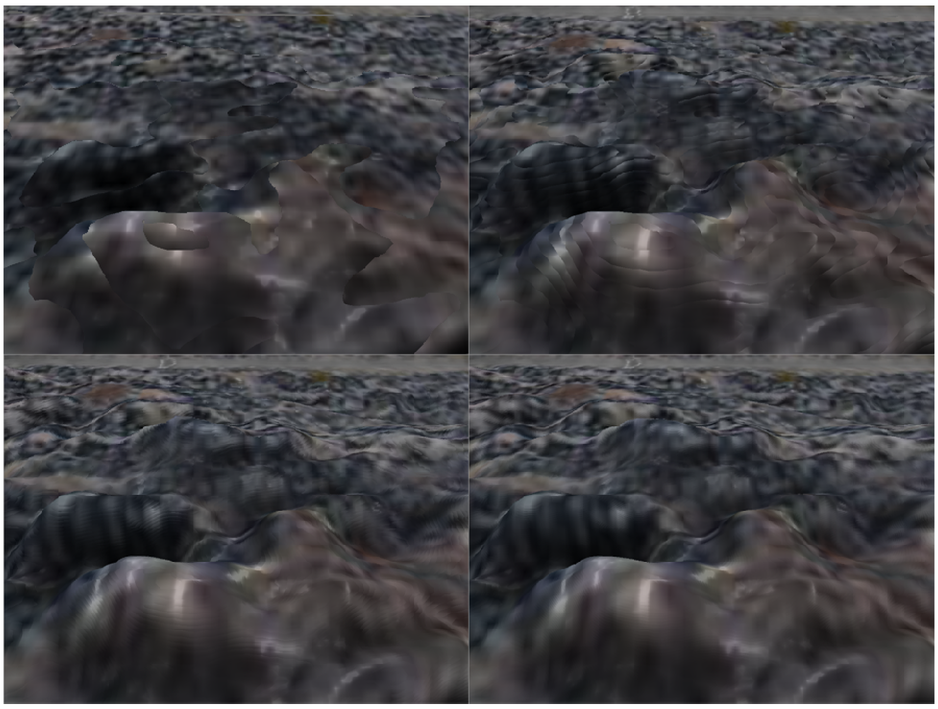


Figure 21: Parallax mapping with different number of steps. (First - Second)
Top left: 4 – 2
Top right: 8 – 4
Bottom left: 12 – 6
Bottom right: 16 – 12

To get an understanding of the visual result, below follows some pictures from the different materials used in the simulation compared with how things might look in real work environments.

The final stone surface is a combination of parallax occlusion mapping, normal mapping and soft shadows. The resulting stone surface looks fairly realistic as can be seen below.

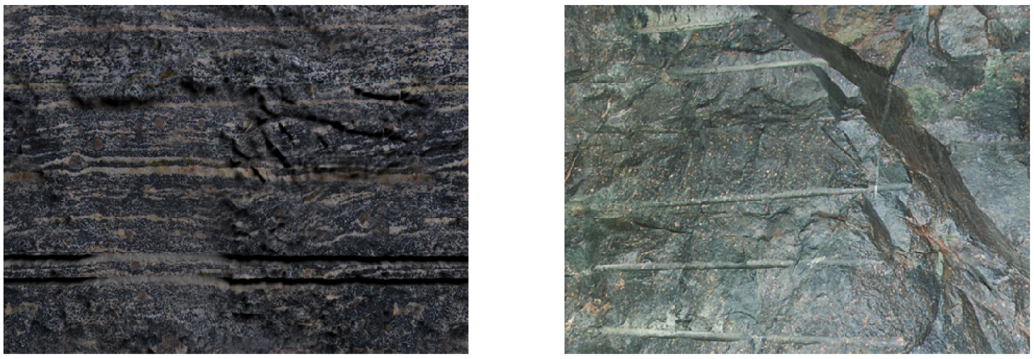


Figure 22: Left, rendered stone. Right, photograph of a tunnel wall (Source: SKB-Äspölaboratoriet).

Visualization of concrete uses the same techniques as the stone. This produces a smooth concrete surface that has many of the characteristics of real concrete as can be seen in the comparison below.

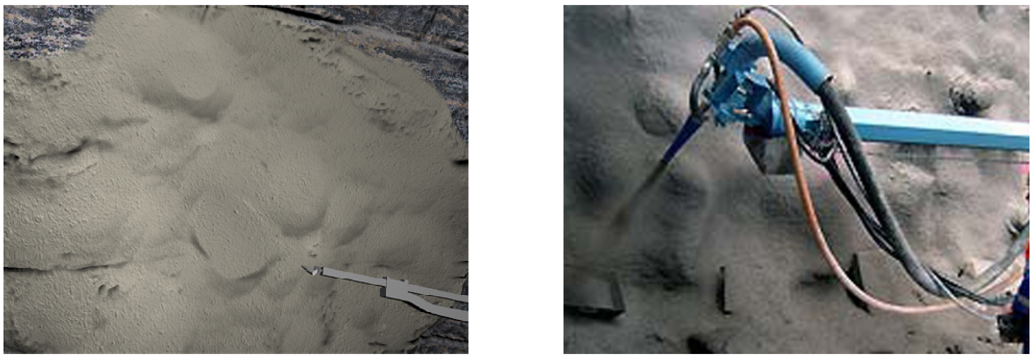


Figure 23: Left, rendered concrete. Right, real construction site. (Source: Meyco)

Overall, parallax occlusion mapping is a technique that works very well to accurately simulate uneven surfaces. However, there have been some problems associated with this technique. One problem with parallax occlusion mapping is that it is computationally expensive. To counter this, it is only used for pixels that are within a certain distance from the camera. When this is done, a noticeable seam is created where the surface toggles from using parallax occlusion mapping to not using it. To prevent this, a region was introduced over the seam where the final pixel color is a blend between two different color values. The first color value is calculated with parallax occlusion mapping and the second without. This works well in the way that it can totally eliminate the seam. However, this means that computationally heavy shader code is run more than once for these pixels which causes a slight drop in performance.

As the tunnel geometry is divided into segments with its own concrete data texture, there are some problems with the parallax effect close to the edges of the segments. More specifically, problems arise when texture values are needed from coordinates that lie outside the current segment. This could be solved by passing concrete data textures from all neighboring segments to the shader. Although potentially solving the problem, it would decrease performance and increase the complexity of the shader. Instead, the concrete data texture uses mirroring as wrapping mode. This produces an acceptable result and eliminates most artifacts this problem introduces. Some artifacts, in the form of edges between segments, are sometimes visible as can be seen in the picture below.

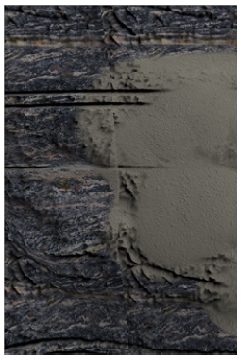


Figure 24: Tunnel segment edge.

5.2 Particles

As the high pressure shotcrete ray hits a surface, mist and smoke form as a bi-product. This effect is simulated using a particle system and the soft particles technique. Since this technique does not produce edges or artifacts, mist particles can be large, slowly moving objects of small quantity which put little strain on the system. Currently there can be a maximum of 10 mist particles alive at a time which is enough to produce the desired effect. The shotcrete ray, on the other hand, needs to use many more particles to produce a realistic result. After testing different combinations of values, it was decided to keep a maximum of 5000 ray particles alive in the simulation at a time. The issue of simulating rebounding particles has not been solved mostly due to lack of knowledge of how rebound looks and behaves in reality.

Below, a comparison between the simulated shotcrete ray and real shotcrete rays can be seen.

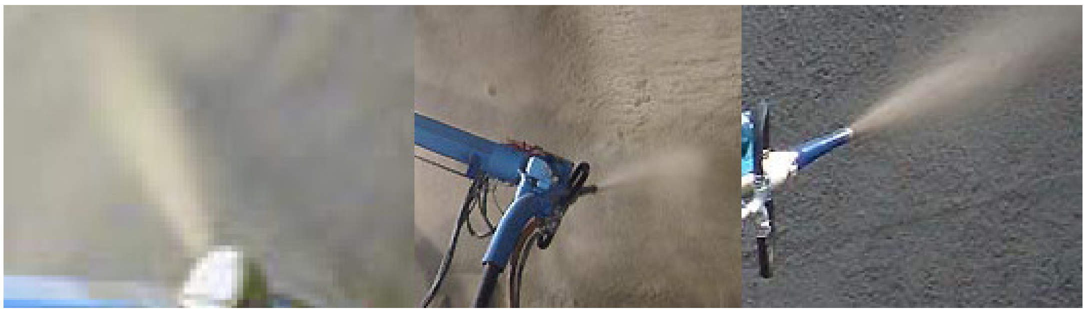


Figure 25: Real shotcrete rays. (Source: Meyco)

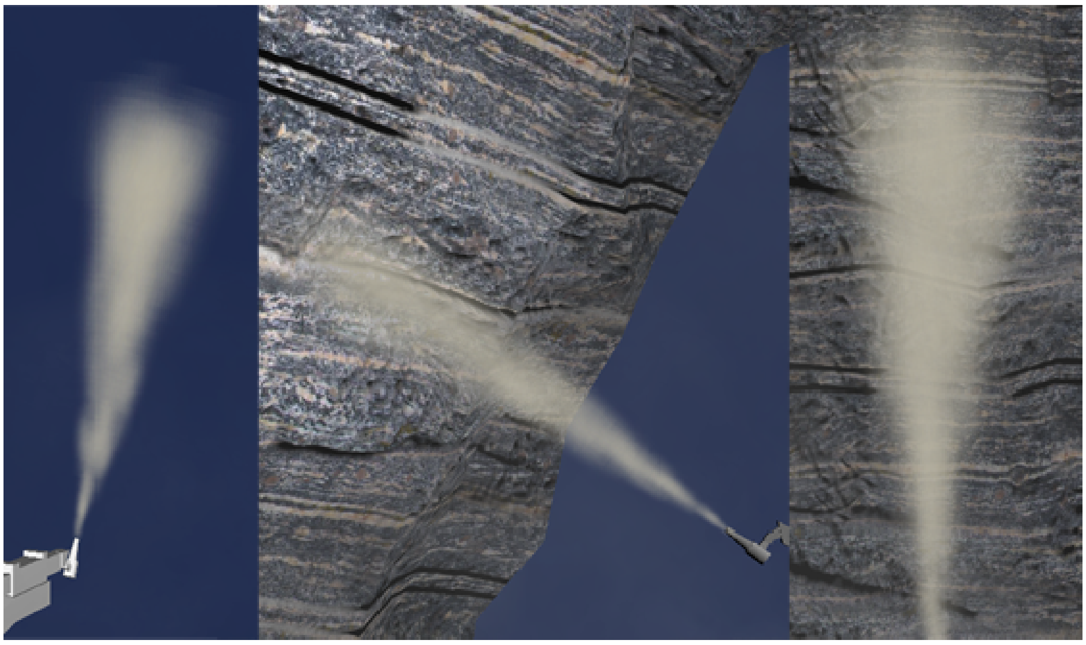


Figure 26: Rendered shotcrete rays.

5.3 Simulation

Currently, the simulation does not use a real robot model. Instead, a temporary approximated model is used. This model is a sufficiently accurate representation to be of use in the current prototype.

Even though the adhesion system does not perform a totally correct physical calculation, the approximation works very well in giving a satisfying result. The approximation is built on the basic principles of concrete adhesion and it gives a result quite close to what one would expect of real concrete. This makes it suitable for this prototype as well as a viable option for future simulations.

The original plan was to visit a real work site to experience the environment first hand. Regrettably, no such field study was possible since these kinds of work sites are quite rare and not always easily accessible. This shortcoming made it harder to grasp the concept of shotcrete reinforcement in its entirety. Instead, the primary source of information was videos and pictures.

Overall the simulation performs well and the parts of the simulator that have been implemented both look and behave as one would expect. The prototype has been shown to company representatives from BESAB, which are experienced in shotcrete operations. The feedback received has been positive and this is an indication that the simulator prototype is on the right track on the road to a proper training simulator.

5.4 Performance

The most computationally heavy subsystem of the simulation is the rendering of the tunnel wall and the concrete layer on top of it. Since most of the computations in this step are done through vertex and fragment shader, this puts the heaviest load of computation on the graphics card. This is good news since it is indeed to this purpose it should be used. The shaders rely quite heavily on dynamic branching and iterations, features that were not supported on earlier graphics cards. These features were first introduced in Shader Model (SM) 3.0 (31) and the shader requires graphics cards to support this in hardware. SM 3.0 was first introduced for the GeForce 6 series (or equivalent) and this could be considered the absolute minimum hardware requirement on which our simulation can be run. Although, according to our tests using a GeForce 6800 card, the frame rate in these tests could barely be considered interactive and the conclusion to this is that this card is not sufficient to run the simulation. Running the program on a machine using GeForce 8800 GTX significantly improves performance and runs the simulation at an acceptable frame rate. It is recommended to use an equivalent graphics card or a card from later series.

Overall the simulation does not require much of the CPU. There are only two parts of the simulation that currently take noticeable CPU time. First, updates of the height textures and sampling of these for statistical purposes are done purely on the CPU. Secondly, the particle systems are currently handled by the CPU. This can place quite a load on the CPU but according to tests made, a processor with a clock speed of 2 Ghz is more than sufficient to handle this.

The memory footprint of the program is low. Most of the memory consumption is due to saving textures with concrete data. The total memory requirement in the final prototype is just barely over 100Mb according to test runs. This is largely dependent on the size of the tunnel or, more importantly, how many segments the tunnel is divided into. As each tunnel segment has its own concrete data texture the memory requirements increase with each added segment.

6 Future Work

This section details future implementation needs and considerations that need to be done, should the project be further developed. Considered are implementation requirements both when looking at the scope of this thesis as well as requirements for a full scale simulator.

6.1 Thesis scope

In the scope of the thesis project there are still things that need to be updated, implemented and considered should the simulator be used in a real setting. Improvements to the adhesion system, a usable interface and extended educational tools are some features that are important in a production environment.

6.1.1 Environment and Equipment

To increase the realism and believability as well as the potential educational quality of the simulator, it would be desirable to have computerized models of tunnels loaded. If such opportunities exist, operating in a model of a real work site could potentially be used as a reference point before starting the shotcrete process on site, saving both time and money.

An additional improvement would be to use a factual, proper model of a shotcrete robot. As the current model is only a crude approximation, it does not operate in the full extent to which a real robot does. For the sake of immersion, having virtual replicas of real equipment would be a certain improvement.

6.1.2 Educational Tools

This thesis project's main goal has been to ascertain the possibility building an educational shotcrete system. As such, not much time has been spent producing scenarios or mission for students to undergo. This would likely be an important addition to a complete system. Such missions might include to apply concrete to an area

- With a certain margin of error in terms of deviation from a nominal height.
- As quickly as possible.
- Using a maximum amount of concrete.
- Only producing a certain amount of rebound.

If specific training scenarios like the above are desired is not known at the time but the possibility to include them in a future simulator exists.

6.1.3 Adhesion system improvements

A more accurate model for calculating adhesion could be of great use as well. Realism is important in a simulator, which includes that of physics. The current implementation uses a very simple approximation which might not be considered good enough. Numerical approaches, such as the Distinct Element Model, that can produce very realistic results, can potentially be used when computer hardware has evolved to a high enough level. If this process is suitable for parallelization it might be possible to utilize the pure computational strength of recent graphics hardware to compute these numerical calculations in real-time using, for example, CUDA (32).

6.1.4 Graphical Improvements

While the result of the graphical parts of the application is acceptable, there is also room for improvements on certain points. The particle systems currently reside on the CPU side of the application which is inefficient since a lot of data has to be pushed to the GPU each frame. Because of this, the simulation could see improvements in frame rate by moving the management of the particle systems to reside entirely on the GPU. This is possible and efficient on recent graphics hardware which supports geometry shaders and stream out functionality.

Another graphical aspect of the simulation that needs improvement is that objects should be able to cast shadows. At the moment the robot does not cast shadows in the world and because of this it becomes harder for users to estimate where the robot is positioned. Without shadows it is harder for humans to determine spatial relationships between objects. Some form of shadows will need to be implemented in a final version of the simulator.

6.1.5 Interface and usability

For the simulator to be of any use in a production environment there is a lot of work to be done on the user interface and the overall usability of the system. Currently, many features are hard to activate and gain access to, something that needs to be addressed by the use of menus, buttons and informational displays. A graphical user interface should also allow users to control the different viewing modes, restart the simulation and perform other tasks.

6.1.6 Performance

The simulation currently runs at a decent frame rate in all situations on a standard PC. It is important to notice that not much time have been spent on trying to optimize the different parts of the simulator. Because of this there is likely room for improved performance both for the code running on the CPU as well as for the shaders running on the GPU.

6.2 High-end simulator scope

To reach the ultimate goal of a full scale simulator there are many things that need to be added or improved. The most important ones are adding support for real robot remote controls and visualization on some sort of VR or CAVE environment.

During shotcrete training, one of the goals is for the driver to become familiar with the controls and how the robot responds. In this first version of the simulator the user have a simple game pad with which to control the robot. While this can be used as a first step in the training process, the need for real control equipment is paramount if the simulator should be of proper use. For this to be realized, the first step is obviously to acquire a proper control device and have it connected to the simulation. If multiple robot types are to be supported, several different control units are likely needed. This process could potentially be costly both in terms of hardware and on the side of software development.

Another challenge is to migrate the simulation from a system with a regular computer screen to an advanced high-end visualization system. There exist many systems for this purpose, each with their own capabilities and requirements. Stereoscopic imaging can be used in order to achieve an increased illusion of depth when viewing an inherently two-dimensional image. This is often accomplished by combining a multiply projected image with a pair of stereoscopic (polarized) glasses.

A CAVE environment features an enclosed room (usually of cubic form), where projectors are directed at the walls of the room, potentially creating an immersive experience. This can give the user the experience of enhanced reality where she can actually walk (albeit quite limited) around and turn around in different directions to get a different look of the virtual environment. This environment can also be used in combination with a pair of stereoscopic glasses.

Research need to be made in order to determine what system would be appropriate for a shotcrete simulation. Depending on what system is used it might impact the simulation in different ways. The basic rendering system will most likely need little to no change in most cases. On the other hand, other areas might need to be reworked. For example, the method by which the operator moves around the environment might not be suitable for all systems. Requirements and potential of different visualization systems differ from each case, making it hard to speculate what might need to be changed, if anything.

Versatility in education may also be important, in which case a need for different kinds of robots is desired. Different kinds of robots are indeed possible implementations in a future system. These could likely be created in-house, from detailed blueprints or specifications, assuming that such are available. Another possibility is to acquire complete 3D models form individual manufacturers. The downside of this would likely be that these would differ in terms of detail and format.

Another form of versatility would be to enable the use of an array of different work sites. Such places could be tunnels of different shapes and sizes, construction sites and other locales. It is possible that the addition of multiple locations would enhance the quality of the educational experience. This way, operators could run the simulator with an environment similar to that in which they are scheduled to work, for example. Different environments could also feature various types of surfaces, having dissimilar conditions for adhesion.

7 Conclusion

During this thesis project a prototype of a shotcrete simulation for educational purposes has been produced. This simulator fulfills the goals stated in the project specification as well as the requirements for a simple simulator as described in (1).

The simulator prototype uses parallax occlusion mapping as the primary method of visualizing stone and concrete materials in a realistic manner. It also shows that simple approximations of concrete adhesion are sufficient to provide a pleasant experience. Overall, the graphical quality of the simulator can be considered to be realistic enough to be of use in education of shotcrete robot operators.

The work here shows that it is indeed viable to build a full scale simulation environment for education of shotcrete robot operators as described in (1). Further work is needed to reach this goal but the prototype produced in this project is a first step in this direction.

8 References

1. **B. Westerdahl, M. Johansson, M. Roupé.** *Simulator för träning av robotförare vid sprutbetongsförstärkning, Förstudie.* Göteborg : Visualiseringsstudion Chalmers, SveBeFo Rapport K27, 2007.

2. **U. C. Puri I, T. Uomoto.** *Numerical modeling - A new tool for understanding shotcrete.* Tokyo : Dept. of Civil Engineering, Institute of Industrial Science - The University of Tokyo, 1998.

3. **BESAB.** BESAB. [Online] [Cited: 06 18, 2009.] http://www.besab.com/.

4. **C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon and J. C. Hart.** *The CAVE: Audio Visual Experience Automatic Virtual Environment.* s.l. : Communications of the ACM, vol. 35(6), 1992.

5. **W. T. Reeves.** *Particle systems - a technique for modeling a class of fuzzy objects.* s.l. : Association for Computing Machinery, 1983.

6. **OSG Community.** osg. *OpenSceneGraph.* [Online] OSG Community, 2007. [Cited: 06 17, 2009.] http://www.openscenegraph.org.

7. **M. Segal, K. Akeley.** *The OpenGL Graphics System: A Specification (Version 2.1).* Mountain View, CA. : Silicon Graphics, Inc., 2006.

8. **J. Kessenich, D. Baldwin, R. Rost.** *The OpenGL Shading Language.* Madison, Alabama : 3Dlabs, Inc. Ltd., 2006.

9. **P. S. Heckbert.** *Survey of Texture Mapping.* s.l. : IEEE Computer Graphics and Applications, 1986.

10. **W. Heidrich, H-P. Seidel.** *Realistic, Hardware-accelerated Shading and Lighting.* s.l. : SIGGRAPH Proceedings, 1999.

11. **Y. Kryachko.** *Using Vertex Texture Displacement for Realistic Water Rendering.* s.l. : Maddox Games.

12. **T. Jenks.** Terrain Mesh Displacement using Vertex Textures. *Terrain Mesh Displacement using Vertex Textures.* [Online] October 2005. [Cited: May 29, 2009.] http://www.jenkz.org/articles/vertextex.htm.

13. **T. KANEKO, T. TAKAHEI, M. INAMI, N. KAWAKAMI,Y. YANAGIDA, T. MAEDA, S. TACHII.** *Detailed Shape Representation with Parallax Mapping.* Tokyo : School of Information Science and Technology, The University of Tokyo, 2001.

14. **W. Donnelly, University of Waterloo.** *Per-Pixel displacement mapping with distance functions.* s.l. : Nvidia Corporation, 2005.

15. **N. Tatarchuk, ATI Research.** *Practical parallax occlusion mapping with approximate soft shadows for detailed surface rendering.* s.l. : Association for Computing Machinery, 2006.

16. **M. McGuire, M. McGuire.** Steep parallax mapping. [Online] 2005.

17. **R. Geiss, M. Thompson.** *Nvidia demo: Cascades.* s.l. : Nvidia Corporation, 2006.

18. **Melbye, T.** *Sprayed Concrete for Rock Support.* s.l. : MBT International Underground Construction Group, 1994.

19. **P. Vedin.** *Sprutbetongsförstärkning, Förslag till förbättringar i produktionskedjan.* Luleå : Luleå Tekniska Universitet, Institutionen för sammhällsbyggnad, 2006.

20. **L. Jinga, J.A. Hudson.** *CivilZone review paper, Numerical methods in rock mechanics.* Welwyn Garden City, AL8 6SG, UK : Royal Institute of Technology, Imperial College and Rock Engineering Consultants, 7 The Quadrangle, 2002.

21. **T. B. Phong.** *Illumination for Computer Generated Pictures.* s.l. : Communications of the ACM, vol 18, no 6, 1975.

22. **D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, S. Worley.** *Texturing and modeling, a procedural approach.*

23. **P. Mamassian, D. C. Knill, D. Kersten.** *The Perception of Cast Shadow.* s.l. : Trends in Cognitive Science, Elsevier Sciences LTD., 1998.

24. **T. Ilmonen, T. Takala, J. Laitine.** *Soft Edges and Burning Things: Enhanced Real-Time Rendering of Particle Systems.* s.l. : Helsinki Univ. of Technology, Telecommunications Software and Multimedia Laboratory.

25. **T. Lorach.** *Soft Particles.* s.l. : Nvidia Corporation, 2007.

26. **Y. Uralsky.** *Efficient Soft-Edged Shadows Using Pixel Shader Branching.* s.l. : Nvidia Corporation.

27. **Logitech.** Dual Action Gamepad. *Logitech.* [Online] 2009. [Cited: May 29, 2009.] http://www.logitech.com/index.cfm/gaming/pc_gaming/gamepads/devices/288&cl=gb,en.

28. **P. Castaneda.** Object Oriented Input System. *Wrecked Games.* [Online] Wrecked Games, 2005. [Cited: 06 17, 2009.] http://www.wreckedgames.com/forum/index.php/board,6.0.html.

29. **Microsoft Corporation.** DirectInput. *MSDN Library.* [Online] Microsoft Corporation. [Cited: 06 17, 2009.]

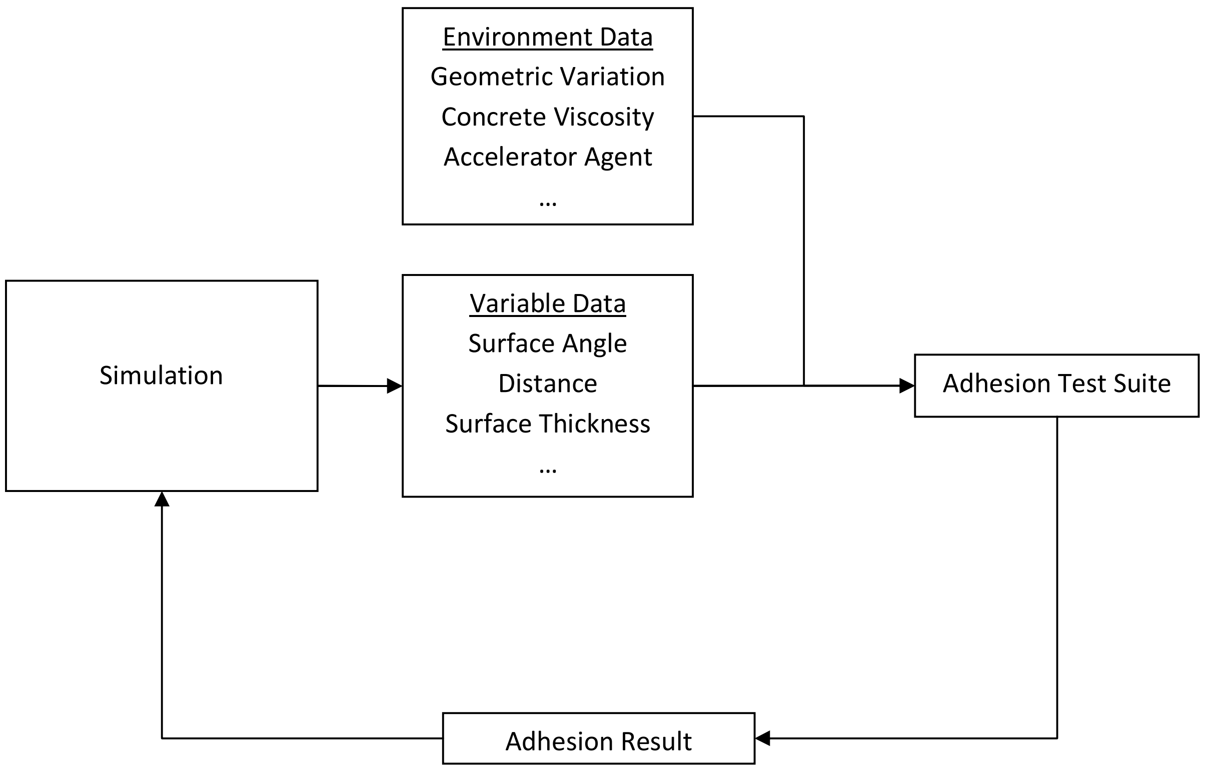
30. **BASF, Meyco.** Meyco Potenza. *Meyco - Expanding Horizons Underground.* [Online] BASF, Meyco. [Cited: 06 17, 2009.] http://www.meyco.basf.com/en/meyco_solutions/equipment/equipment-range/Pages/potenza.aspx.

31. **A. Rege.** *Shader Model 3.0.* s.l. : Nvidia Corporation, 2004.

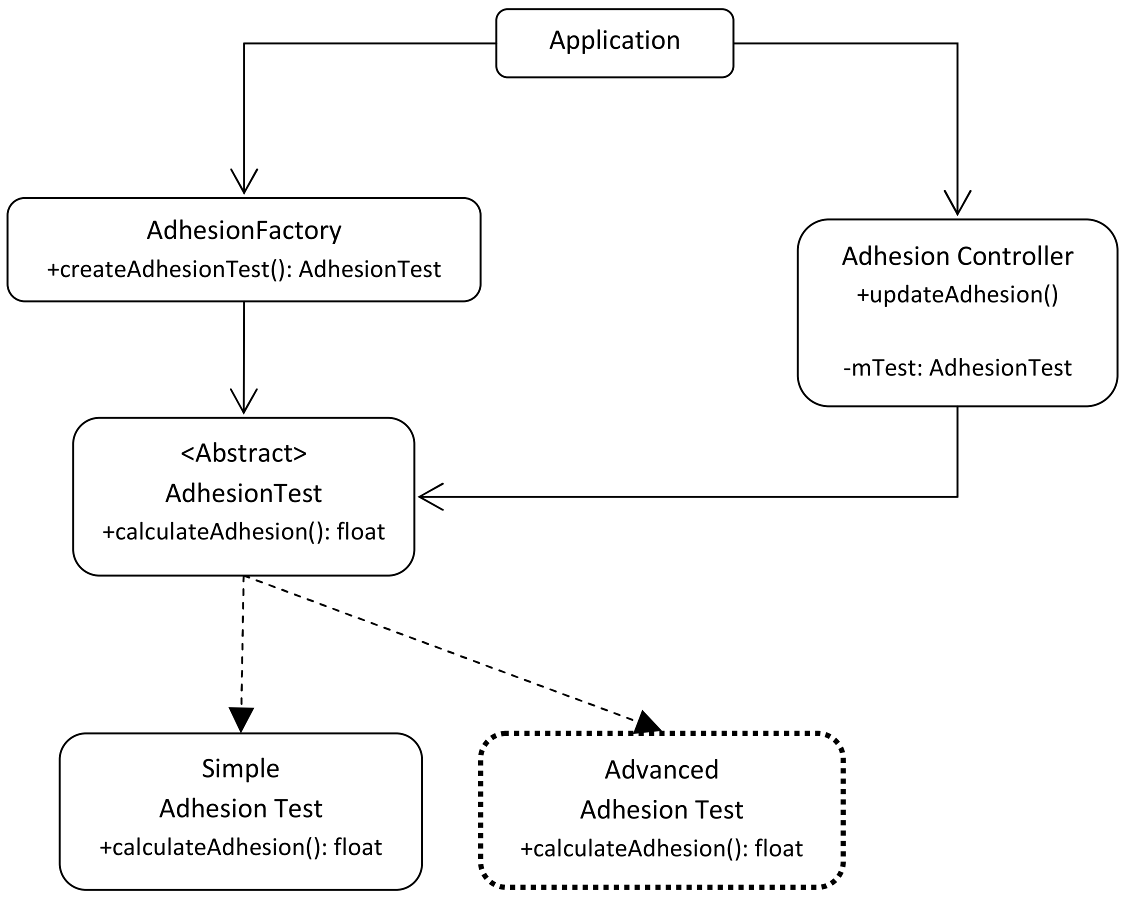
32. **Nvidia Corporation.** *NVIDIA CUDA - Programming Guide 2.2.* Santa Clara, CA : Nvidia Corporation, 2009.

Appendix A – Implementation Diagrams

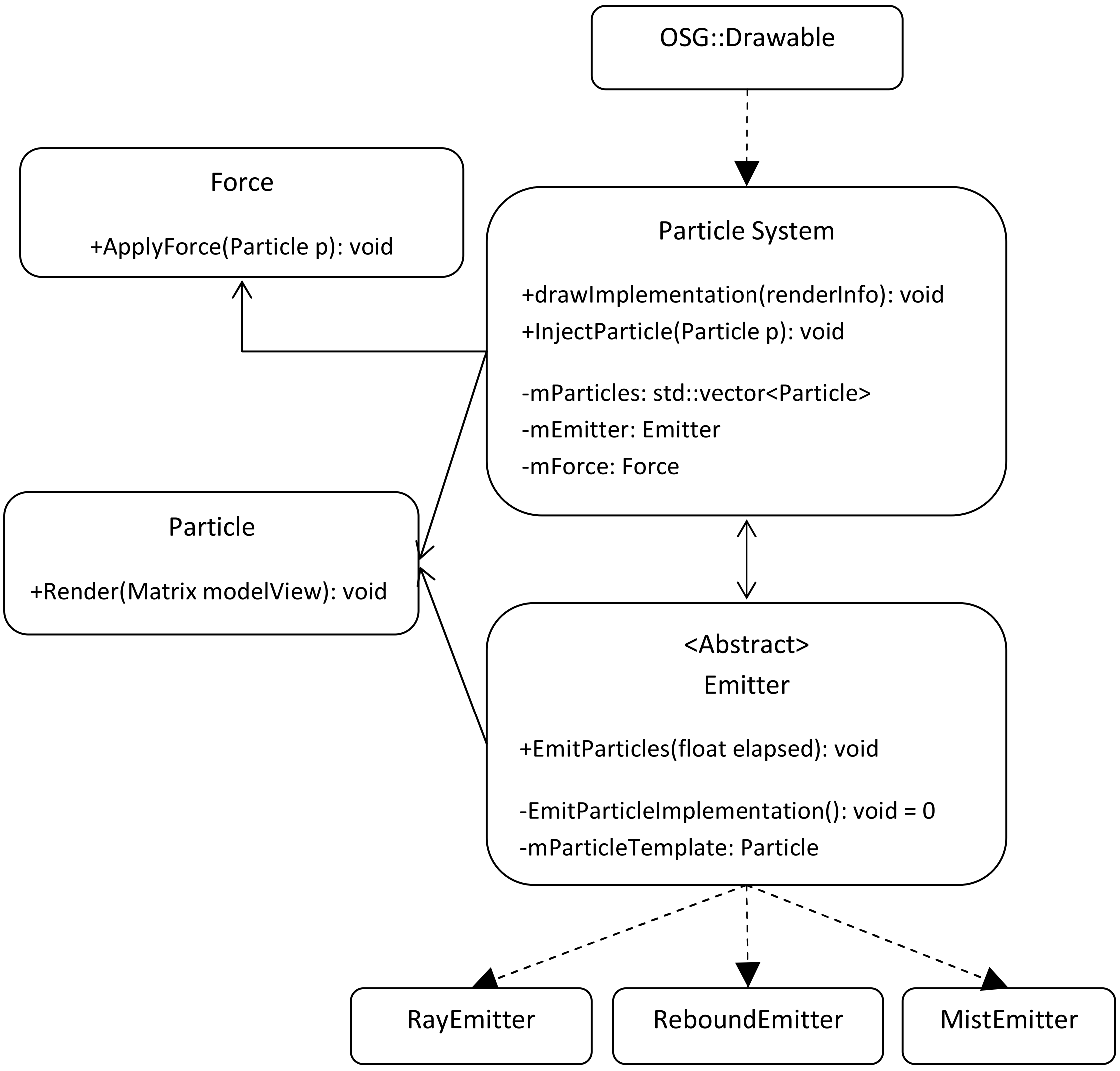
1. Adhesion Overview



2. Adhesion System Overview



3. Particle System Overview


OSG::Drawable
Force
+ApplyForce(Particle p): void
Particle System
+drawImplementation(renderInfo): void
+InjectParticle(Particle p): void
-mParticles: std::vector<Particle>
-mEmitter: Emitter
-mForce: Force
Particle
+Render(Matrix modelView): void
<Abstract>
Emitter
+EmitParticles(float elapsed): void
-EmitParticleImplementation(): void = 0
-mParticleTemplate: Particle
RayEmitter
ReboundEmitter
MistEmitter