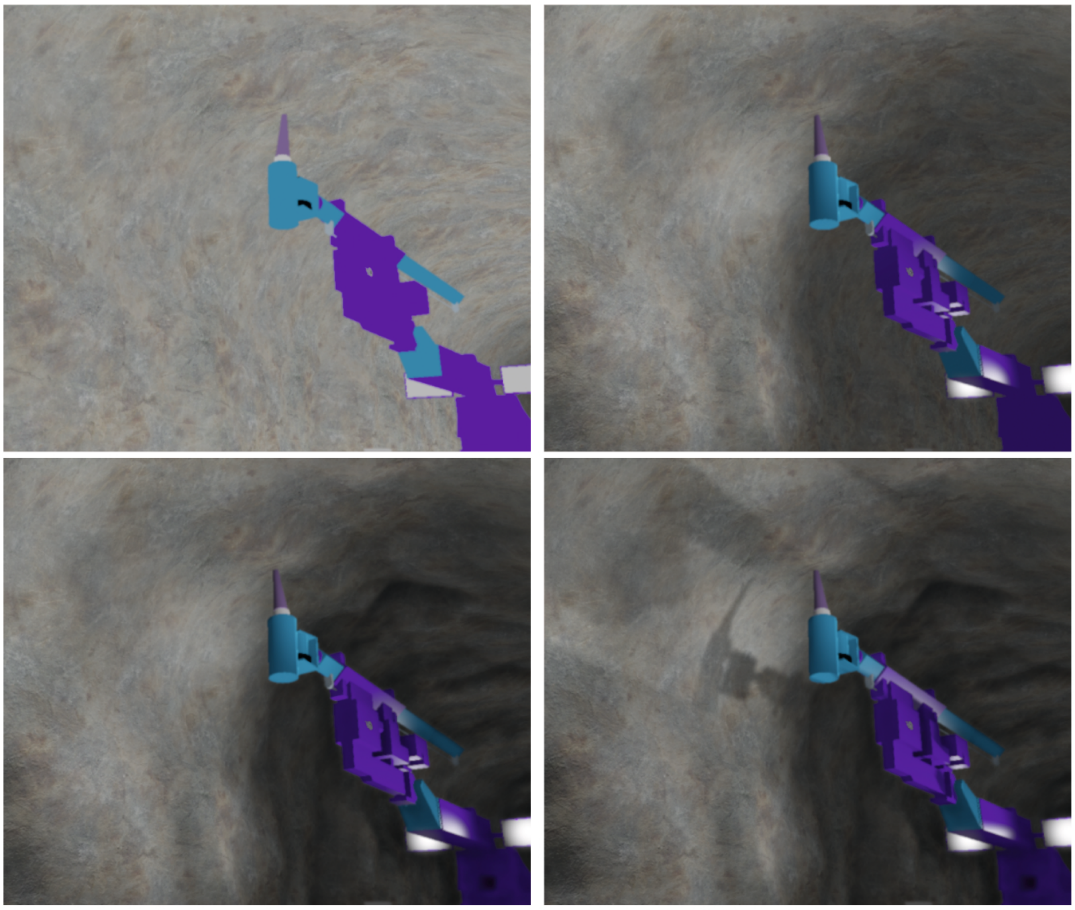
BeFo
STIFTELSEN BERGTEKNISK FORSKNING
ROCK ENGINEERING RESEARCH FOUNDATION



SIMULATOR FOR TRAINING ROBOT OPERATORS

– Virtual education in shotcrete operation

Petter Börjesson
Mattias Thell

BeFo Report 111

Cover Figure:
Scene rendered with different effects. From top left to bottom right.
1. Only ambient lighting. 2. Added directional lighting. 3. Added SSAO.
4. Added shadow maps. (Figure 12 page 20).

ROCK ENGINEERING RESEARCH FOUNDATION
STIFTELSEN BERGTEKNISK FORSKNING

SIMULATOR FOR TRAINING ROBOT OPERATORS

– Virtual education in shotcrete operation

Simulator för träning av robotförare
– Virtuell utbildning för sprutbetongsoperatörer

Petter Börjesson
Mattias Thell
Chalmers University of Technology

BeFo Report 111
Stockholm 2011
ISSN 1104 – 1773
ISRN BEFO-R—111—SE

Preface

Operator training is essential for handling advanced machinery and vehicles. For many years training simulators have been used for pilots, and later for drivers of heavy vehicles like loaders, rock drilling machines, wood harvesting machines etc. Experience demonstrates considerable benefits from simulator training in decrease of damage, accidents and increased efficiency. Today development and manufacturing of training simulators for all kinds of advanced machinery has become an individual business field.

However, up to now training simulators for shotcrete robot operators has not been available. Shotcrete is used for rock reinforcement in tunnels, mines and other applications. Shotcrete has demonstrated being more cost efficient compared to concrete lining in many cases especially in hard rock environment. Shotcrete application is a handicraft a demand skilled robot operators. Training in real environment implies large costs and unsecure circumstances. Consequently shotcrete operators would benefit from simulator training before operating in the real environment.

Training of shotcrete robot operators comprise both handling the equipment and to judge how this needs to be done based on intermediate parameters for the concrete to stick to the surface to largest possible extent. Just like simulator training programs shotcrete robot simulator training include exercises for evaluating operator level of skill and performance.

This project has developed a simulator software program usable to educate shotcrete robot operators. The idea was initiated by Tommy Ellison at BESAB and the development was conducted by a project group at Chalmers Visualization Technology by Petter Börjesson and Mattias Thell supervised by Börje Westerdahl. Financial support was provided by BeFo/Formas and SBUF. The project will continue in a commercial phase under the name of Edvirt in cooperation with Encubator AB at Chalmers University of Technology.

Stockholm in December 2011

Mikael Hellsten

Forskningsrådet för miljö, areella näringar
och samhällsbyggande, Formas

SAMMANFATTNING

Sprutbetong används för att förstärka bergstrukturer under konstruktionen av tunnlar, gruvor och många andra projekt. Betongen slungas i hög hastighet på ytor med hjälp av tryckluft. För att hjälpa denna process används ofta robotutrustning. Dessa robotar styrs manuellt av operatörer som är ansvariga för att utföra förstärkningsarbetet enligt specifika instruktioner och regler. Kvalitetskraven för denna typ av arbete är mycket höga och operatörerna måste vara välutbildade både i teorin bakom arbetet och i användandet av utrustningen. I dagsläget utförs den största delen av utbildningen ute på arbetsplatser. På grund av nya operatörers oerfarenhet leder detta till ökade kostnader, osäkra arbetsförhållanden och ökad tidsåtgång.

För att minska problemen med helt oerfarna operatörer ute på riktiga platser har detta projektet gått ut på att utveckla en mjukvara som ett alternativ till praktisk utbildning. I mjukvaran kan framtida sprutbetongoperatörer virtuellt kan träna på de viktigaste färdigheterna som krävs för att korrekt styra en sprutbetongrobot. Mjukvaran har utvecklats i programmeringsspråket C++ och använder sig av realtidsgrafikramverket OpenGL och består av en simuleringsmotor som kan återskapa arbetsmiljön för en sprutbetongoperatör samt en grafikmotor som kan visuallisera miljön i 3D på en vanlig bildskärm eller i sterioskopisk 3D med hjälp av glasögon i aktiv stereo. På detta sätt kan man med en vanlig dator erbjuda operatörer en inblick i en typisk arbetsmiljö där man kan styra en robot med realistiska kontroller och få omedelbar återföring på hur robot och betong beter sig i olika sammanhang. Simuleringsmjukvaran består av följande komponenter.

Sprutbetongroboten och den tillhörande kontrollen är en av de viktigaste aspekterna i simuleringsmjukvaran och är en av de punkterna som det jobbats med för att få en så realistik upplevelse som möjligt. Först och främst så har ett protokoll utvecklats vilket gör det möjligt att koppla in riktiga robotkontroller till simulatorn. Detta innebär att robotar i simulatorn kan styras på samma vis precis som de hade styrts i verkligheten. Robotarna i sig importeras till simulatorn via 3D-modelleringsprogrammet Maya. En utökining till Maya har skrivits för att konstruera och exportera robotmodeller vilket även ger utvecklingsmöjligheter i form av att man kan lägga till fler robotmodeller på ett enkelt sätt i framtiden. Den robotmodellen som finns i den nuvarande mjukvaran är en testmodell som representerar en riktig robot. Roboten har en leduppsättning som liknar moderna robotmodeller på dagens marknad och styrs på samma sätt via den riktiga kontrollen.

En andra viktig punkt för få en bra träningsupplevelse är att miljöerna man jobbar i återspeglar förutsättningarna som finns i en verklig arbetsmiljö. För att åstadkomma detta visualiseras arbetsmiljön så realistiskt som möjligt med hjälp av till exempel ljussättning och skuggor som grafikmotorn hanterar. Den geometriska information om arbetsiljöer är också viktig då ytans utformning är något en operator måste hålla under uppsikt för att kunna göra ett bra arbete. För att få så realistisk miljö som möjligt har det arbetats med att omvandla punktmoln från laserscaningar av riktiga tunnlar till geometri som kan laddas in av simulatorn.

Den tredje punkten för att kunna träna operatörer är att roboten skall kunna spruta betong som fäster på ytorna i arbetsmiljön. För att åstadkomma detta så simulerar mjukvaran ett betongflöde med hjälp av partikelsystem samt beräknar hur mycket av betongen som fastnar på ytan man siktar på. Hur mycket betong som fastnar bestäms av vidhäftning. Vidhäftningen av betong påverkas av många parametrar som till exempel lufttrycket som betongen sprutas med, betongblandningen och hur hårt underlaget man sprutar på är. De två parametrar som påverkar vidhäftningen mest är avståndet från munstycket till ytan samt att munstycket har en rät vinkel mot ytan. Hur bra vidhäftning man har avgör hur mycket av materialet som används effektivt och på grund av detta är avstånd och vinkel en av sakerna som simulatorn fokuserar på.

För att veta hur bra resultat en operatör har beräknar och sparar simulatorn statistik över en mängd olika parametrar. Dessa kan operatören se antingen i realtid under träningspassen eller i efterhand efter avslutad övning. Några av parametrarna som sparas är betongvolymen operatören har använt. Det sparas även hur mycket av detta som är effektivt använt och hur mycket som slösats. Operatören kan även se vilket mönster munstycket har tagit under övningstillfället samt undersöka tjockleken över hela arbetsytan. Med hjälp av statistiken och de visualiseringsverktyg som finns i simulatorn kan operatör eller handledare analysera ytor och betongmängder för att avgöra vad som gick bra och dåligt under övningstillfället.

Simulatorn inkluderar även ett kurssystem i vilket man kan sätta upp övningar som en operatör skall få godkänt på för att klara kurser. Detta system använder sig av statistiken som simulatorn samlar in under ett träningspass och analyserar den automatisk utefter de kriterier som är specifierade för övningen. Operatörer får logga in i systemet med sina användarnamn och systemet kan sedan hålla reda på övningar och kurser som olika användare har klarat. Detta skulle kunna utnytjas i sammband med kurser eller certifieringar i olika sammanhang.

Huvudmålet med mjukvaran är att den ska användas för att ersätta så stor del som möjligt av den praktiska utbildningen för helt oerfarna operatörer. Förmodligen kommer inte praktisk träning kunna ersättas helt då det är svårt att få en simulering såpass verklighetstrogen men kan man till exempel halvera tiden praktisk träning är det väldigt värdefullt. Förväntningen är att mjukvaran och virtuell träning avsevärt kommer att reducera kostnaderna för att utbilda operatörer genom att minska tiden oerfaren personal opererar i riktiga projekt. Detta leder i sin tur leder till mindre materialspill, mindre korrigeringsarbete, kortare tidsåtgång samt att man kan undvika säkerhetsrisker associerade med oerfaren personal.

Nyckelord: sprutbetong, simulering, datorgrafik

SUMMARY

Sprayed concrete, or shotcrete, is a method of applying concrete as a reinforcement agent during construction of tunnels, mines, and many other projects. The concrete is projected pneumatically onto surfaces at high velocities. To aid this process, robotic equipment is used. The robots are manually controlled by operators that are responsible for performing the reinforcement according to regulations and specific instructions. The quality demands on this type of work are very high and the operators need to be well educated in controlling the equipment and understanding established procedures. In the industry today, education is mostly performed on live production sites. Due to the inexperience of beginners this induces extra costs, decreased safety and increased production time.

With the goal of alleviating this problem, in this project, a virtual software system for the education of shotcrete operators has been developed. The software has been developed in the programming language C++, using the real-time graphics framework OpenGL.

The software program is capable of simulating the shotcrete experience and allows users to operate a robot in real time and receive instant visual results. This is performed by operating an authentic control device which is connected to the computer, relaying signals to the virtual robot. Concrete flows from the nozzle of the robot, adhering to the surface. The adhesion itself is dependent on various parameters, such as distance and angle to the surface. Available to the users are a number of options for feedback, both during and after a training session. These include real-time adhesion feedback, visual aids and statistics.

The software is to be used to partially replace real world practical education. This is expected to significantly reduce the costs for educating operators. Moving the education to a virtual setting will also greatly decrease, or completely eliminate, safety concerns during training.

Key words: shotcrete, simulation, computer graphics

Contents

1 Introduction

This project have been carried out at the Department of Structural Engineering, Construction Management, Visualization Technology with the support of Mikael Johansson, Mattias Roupé and Börje Westerdahl.

Tommy Ellison works with shotcrete at BESAB in Göteborg and is the person that came up with the idea and initiated this projekt. Tommy and BESAB have provided valuable industry experience to the projekt.

Ulf Assarsson at the Department of Computer Science and Engineering acted as advisor on computer graphics issues during the development of the software.

In the beginning of the project a number of industry professionals from different Swedish companies were invited to partake in a reference group to provide input and feedback during development. Meetings with this group were held a few times each year and the feedback from these meetings helped the project a lot The following people were invited to partake in the meetings:

-Martin Bergström - Tyréns
Regional chief, west

-Lars O. Ericsson, Chalmers University of Technology
Associate professor at the division of Geology and Geotechnics

-Henrik Eriksson - BESAB
Experienced operator of shotcrete robots.

-Quanhong Feng - 3D MultiInfo 3D Laser Scan Solution AB
Works with laser scanning of tunnels and other construction sites.

-Mikael Hellsten - BeFo
Research director at BeFo

-Per-Erik Josephson, Chalmers University of Technology
Professor at the division of Construction Management.

-Benjamin Krutrök - LKAB
Chief of produktion of shotcrete and concrete at KGS AB.

-Robert Sturk - Skanska
Technical chief.

-Gunilla Teofilusson - CBI Betonginstitutet
Works with education in the use of concrete in different areas.

-Per Vedin - Trafikverket
Rock technician at Trafikverket

-Kjell Windelhed - Trafikverket
Rock technician at Trafikverket

2 Background

2.1 Problem description

In the construction and mining industry, an important aspect of the work is the ability to rapidly and effectively strengthen existing rock surfaces and structural elements using a 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. The concrete flows in a hose and out through a nozzle at the end. This is called sprayed concrete reinforcement, shotcrete reinforcement, or simply shotcrete.

Shotcrete reinforcement is most often performed with the help of a robot and is used during construction and repairs of tunnels, mines, rock shelters, and many other scenarios. The robots are manually controlled via remote by operators that are responsible for performing the reinforcement according to regulations and specific instructions, such as concrete thickness. Also, to achieve sufficiently good results, the nozzle must be kept at an optimal distance from the surface, in the correct angle. This is not an easy task as the working surface is often uneven and fragmented, for example in the case of blasted rock in mines or railway tunnels.

Today in Sweden, there are no educations for shotcrete robot operators supported by the government. All education on the practical aspects of shotcreting is therefore performed internally at companies that do this kind of work. The mining industry is experiencing a growing market and there are extensive plans for new infrastructure projects. This is one reason to believe that demand for shotcrete robot operators will increase in the coming years.

To learn how to operate a shotcrete robot correctly, practical training is essential. Today, this is mostly performed at a live production site with supervision from an experienced operator. This quickly becomes very expensive both due to material waste as well as corrections needed where the necessary quality requirements are not met. Another option available is to put students in a dedicated practice environment. This option has the benefit of not having projects suffer from any mistakes made during training. The downside is that costs are still high as material, equipment and preparation of the environment is expensive.

It takes a lot of practice with a robot before an operator can be considered fully educated. This leads to high costs for the companies and there are no particularly effective training methods available. An alternate method which lowers costs is needed. At BESAB, the idea was born to educate nozzlemen in a virtual environment. This would likely greatly reduce costs by reducing faults, improve safety, decrease installation times, and be a good supplement to practical training.

2.2 Goal

The goal with this project has been to develop a computer simulation that can be used to educate shotcrete operators in a virtual environment. To accomplish this, a computer program has been developed where operators can perform shotcrete reinforcement in a virtual environment, in real time. The aim has been to produce a realistic simulation where the operator can learn to maneuver a shotcrete robot, learn the procedures of concrete application and review different quality aspects of the result. This includes statistics such as surface coverage, concrete thickness and material consumption. The idea is that such a simulator can replace part of the practical training and in this way avoid many of the expensive mistakes new operators do at real work sites.

2.3 Previous work

In 2007, a pre-study was compiled at Chalmers Visualization Technology (B. Westerdahl, 2007). This report showed that it's a reasonable goal to try and produce a virtual training system. By 2009, further development was done as a master's thesis project (P. Börjesson, 2009) at Chalmers. During this project a prototype was developed which includes virtual tunnel environments, real time concrete updates and robot control. The prototype produced here showed that it would definitely be possible to produce a training simulator for shotcrete robot operators.

2.4 Introduction to Computer Graphics

The following chapters can at times become quite technical in nature and much of it revolves around the domain of real-time computer graphics. It is understandable if the concepts can seem far from trivial, and it is the hope of the authors that most people should still be able to follow along. Therefore, this section contains a (very) brief introduction to the topic, along with several terms that is used throughout the rest of the document.

Digital images are constructed of a two-dimensional grid of colored points, or pixels. Each pixel consist of a triplet with three colors; red, green and blue. Each of the three colors is specified with a value ranging from 0 to 255. This means that each pixel is capable of representing about 16.7 million (256^3) different color values.

The most common way of modeling 3D geometry is to use polygons. Due to its simplicity, the most commonly used polygon is the triangle. A triangle consists of three points in space, or vertices, which is the least number of points required to represent a surface in three dimensions. Combining triangles, we can approximate very advanced shapes and this is the way most real-time computer graphics handle geometry. A collection of triangles that represent an object is often called a mesh.

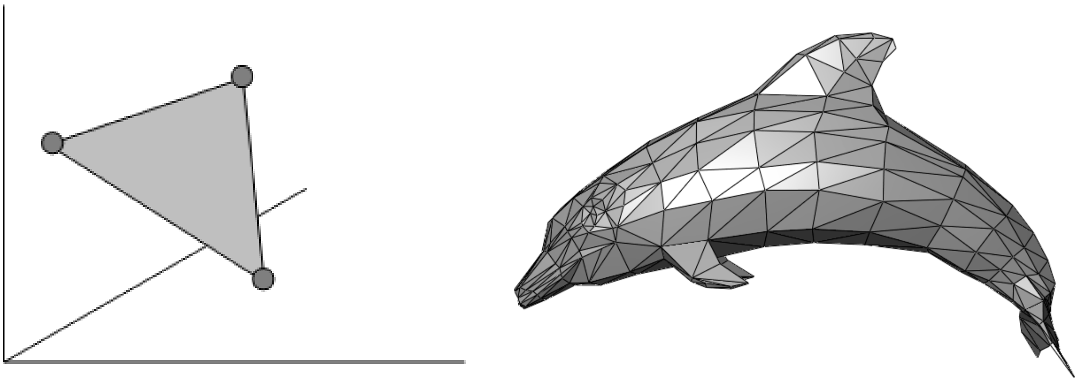


Figure 1: Left, a triangle in 3D space. Right, a 3D mesh built of triangles.

Cameras in 3D function much like its real world counter-parts. It is set up with position, direction, rotation and other properties, such as field of view. Through a process called rasterization, the objects seen by the camera are projected onto pixels on the screen. The process of displaying a 3D scene on a screen is called rendering the scene.

A texture is a color image that can be applied to a mesh in order to give it color and structure. For example, you might apply a texture depicting bricks to a flat mesh to give it the look and feel of a solid wall. Other than color data, a texture can also be use to convey different kinds of information, such as height or depth.

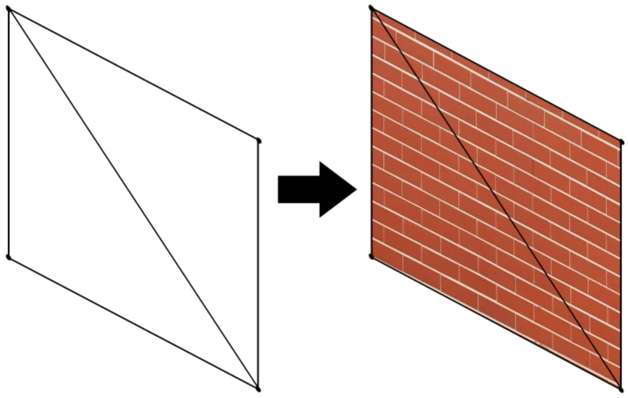


Figure 2: A brick texture applied to a mesh surface.

Lights (and shadows) can be added to enhance the quality of the rendered image. Lights alter the shading of objects. In computer graphics, as well as in classical illustrations, shading refers to the effect of applying different levels of darkness to different parts of an object, relative to its position to the light. In computer graphics, the nature of the interaction between the object and the light is also considered when applying this term. Lighting is dependent on the so called surface normal. A normal describes the direction of the surface relative to the light source. Normals are most often specified per vertex.

Nowadays, the rendering capabilities of a computer are enhanced by specialized hardware, graphics cards. Such a card is equipped with a so called *Graphics Processing Unit* (GPU) that does all the calculation legwork. Computers are also equipped with a *Central Processing Unit* (CPU), which takes care of non graphics related calculations.

3 Implementation

This section describes the implementation details of the simulation software. The different parts of the program are detailed along with considerations and requirements concerning them. The software has not yet a name, but for the sake of brevity, in this and following chapters, the abbreviation SOE (Shotcrete Operator Educator) will be used.

At the start, a rough requirements specification was written, and the major parts of the software were laid out. The requirements for the software were jointly established by the software development team and representatives from the industry. Succinctly, the plan consisted of splitting the project into a number of separable pieces, each responsible for a specific task. This separation of concerns is not only a fundamental theory of practice in computer science, but it also allowed the development team to work on different parts in parallel. All parts of the system were developed iteratively, and evolved throughout the project.

3.1 General Requirements

On the technical side, a number of requirements were established early on. Not pertaining to any particular section of the program, these were constraints that were determined to be important in all aspects of the program.

SOE has, of course, always intended to be a real-time simulation. This means that there must be some constraints on minimum frame rate allowed. It was decided that this limit be set to 60 Hz, or 60 updates per second. At this frame rate, the brain can no longer distinguish between individual frames of images and the result is a smooth viewing experience. In turn, this means that each frame can, at maximum, take 16.6 milliseconds to compute.

The domain requirements for the software were not set in stone, and in fact still aren’t, at the beginning of the project. Should a change in these requirements occur, it was therefore very important that changes to the design be implemented smoothly. This meant that the code needed to be extensible and modular.

3.2 Robot

The prototype featured a functional, although aesthetically displeasing, robot model. While technically correct, this robot model was on par with older robot models and not similar to the modern equipment that is in use today. Therefore, it was decided that a new model would be needed, replicating the functionality that is found in modern robotic equipment. At the same time, a system to more easily create and import robots to the simulator was developed.

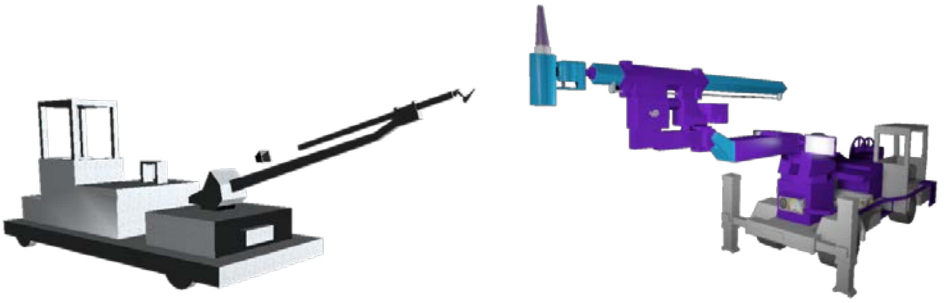


Figure 3: Evolution of the robot model used in the project.

The data needed to specify a complete robot model consists of a few pieces that need to be specified at creation time:

3D mesh and Texture Data

Detail information that defines how the robot looks once in the simulator.

Physics Collision Data

The graphical and physical representation of the robot is separated in order to make data processing more efficient, as seen by the engine. The collision hull consists of a number of low-detail convex shapes.

Joint Connections

The robot arm consists of a number of pieces connected by joints. Each joint can either be rotational or translational and fixed with certain constraints.

Lights

A robot usually has a number of spotlights on its hull.

Nozzle

The position and direction from which the concrete flows.

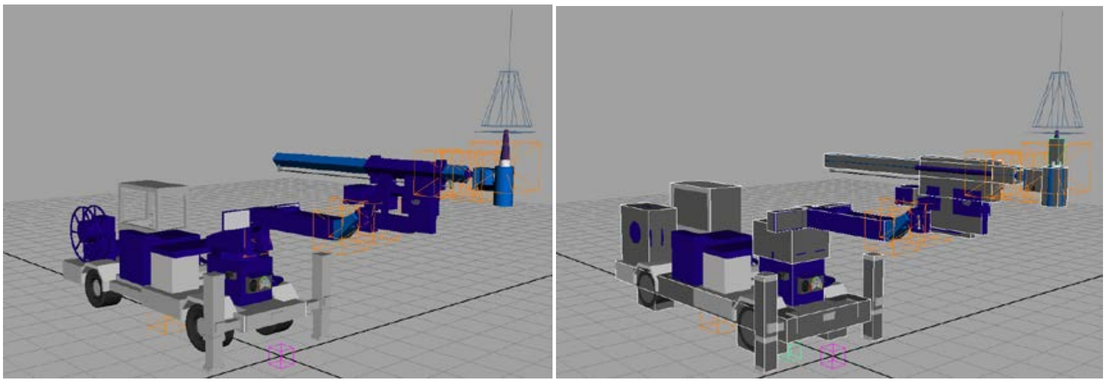


Figure 4: Construction of the robot inside Maya.

The team used a program called Autodesk Maya (Autodesk) to construct and assemble robot models. Out of the box, Maya is a 3D modelling and animation solutions that is widely used in the games and movie industries. Alongside of 3D mesh authoring, it is also capable of loading custom plug-in modules. This capability has been exploited and a custom “robot authoring” plug-in for Maya has been written. This plug-in has allowed the team to specify all the necessary pieces (as seen above) in a single environment. This information is exported to a binary file which is loaded by the simulator program.

3.3 Physics

It was decided that a third party solution should be used for the physical simulation of the robot and its parts. Given the lack of expertise in this area and that there are a number of highly regarded physics libraries available this seemed like a wise choice. It was decided that an open source solution would be the best fit and that the Bullet Physics library (Bullet Physics) would be used. The Bullet library is a mature, widely used project with performance benefits over its competitors. Essentially, such a library takes the physical representation of the robot (that is, the physics collision data, as explained above) and calculates motions of entities as accurately as possible.

At first go, the SOE robot was assembled using Bullet constraint modifiers. These allow rigid bodies to be jointed together and move relative to each other. A shotcrete robot is fundamentally a series of connected parts operating co-operatively. It was discovered that this connectivity posed significant problems for the physics library which struggled to maintain stability of the system. This instability manifests itself as wild jumps and jitters instead of a smooth continuous motions.

The instability itself has its cause in how the computer performs the physics simulations. Rather than exact knowledge, real-time simulations operate on predictions of motion; given an object’s speed, and mass (and other properties) and given some time in the future, it can be calculated where it should be at that time. There are several numerical integrator solutions that can be used for this, with the better ones resulting in smaller magnitudes of error. A more accurate integrator or running enough iterations to produces a stable result induces a heavier load on the computer, so a sufficient middle ground must be taken. Nonetheless, at some rate errors will inevitably creep in to the results, causing instability.

In a sense, a series of connected entities will propagate errors the longer the chain is. The errors will accumulate to a point where the simulation is so unstable that the results are no longer usable. Therefore, the constraint modifier system was foregone in favor of a more direct approach. Instead of using Bullet’s dynamic physics simulation to control the joints, a simpler and more stable solution is now used. This works by letting all joints connect to each other at a static point. Instead of forces being applied in the physics system each joint has an individual force applied to its center of rotation or translation. Each transform is then propagated along the chain of joints, producing a good result of how a robot behaves in reality.

It should be noted that the above described physics simulation has very little to do with the concrete flow and adhesion. These systems are handled separately in a different part of the program.

3.4 Hardware Interface

To be able to control robots in the simulation, some way of connecting the remote control to a PC was needed. KranCom (KranCom), a company based in Göteborg, specializes in this kind of equipment, and they were able to modify an authentic robot control device for our purposes. The physical interface used between robots and control devices has been re-interpreted to connect to a PC environment instead of a robot. Via radio, the device transmits signals about button presses and joysticks movements to a receiver. The receiver is connected to the computer via a serial interface. These serial signals are interpreted by the SOE and results in actions by the robot.

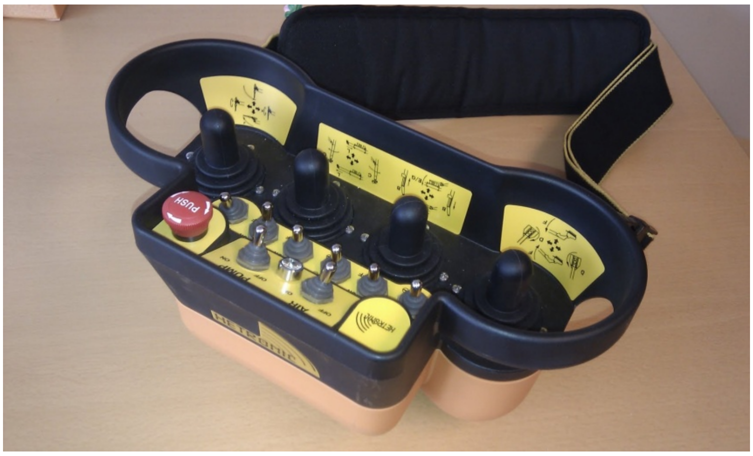


Figure 5: Example of a control used by shotcrete operators.

It was a fairly straightforward procedure to map the signals from the device to events accepted by our software. The receiver sends a constant stream of bytes and these bytes must be interpreted as button presses. Some trial and error testing was necessary to determine which signals corresponded to which buttons.

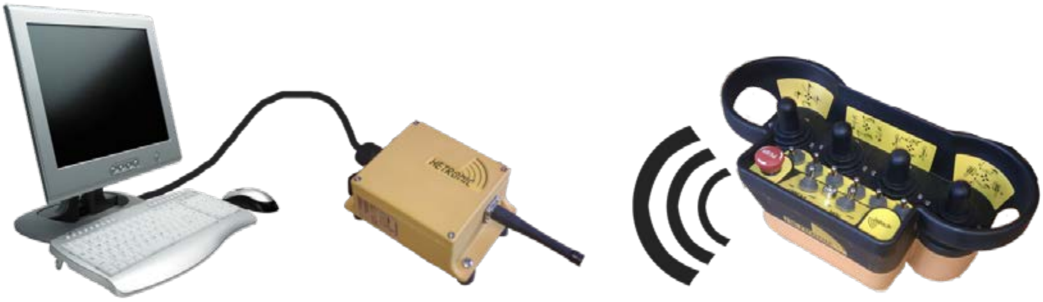


Figure 6: Connection scheme for the equipment.

This software bridge has naturally been written with extensibility in mind and the proprietary control device currently in use can be replaced for another with minimal work. This feature is important in order to support new kinds of hardware, both in terms of robots and control devices.

3.5 Environment

This section will describe how the environments were created, how they are saved and loaded and finally how they are rendered.

3.5.1 Concrete data

Arguably, one of the most important parts of the SOE is the interaction between the concrete and the surface to which it is applied. Technically, this poses some interesting questions and introduces a number of requirements. The concrete needs to interact with the environment by filling cavities, creating extrusions and smoothing the surface. It also needs to be rendered convincingly. Also, making assumptions is difficult because the user can potentially shoot concrete at any point in the environment at any given time. A number of techniques have been considered and tested.

3.5.1.1 Parallax Mapping

Parallax mapping (N. Tatarchuk, 2006) is a texture based technique that was tried during very early versions of SOE to see if it would be useful for visualization of both rock and concrete surfaces. The interesting thing about the method is that it allows for the rendering of increased detail (such as small holes, extrusions and the like) without actually increasing the complexity of geometry. This technique works by using a height map texture applied to the surface and, for each rendered pixel, a ray trace is performed against this height map. During the trace, texture coordinates are offset until the ray hits the surfaces of the height map and this creates the illusion of depth in an inherently flat surface. Parallax mapping works very well for creating more visual detail without increasing geometric complexity and it's easy to scale the resolution of detail by increasing or decreasing the height map resolution. Unfortunately it suffers from a few drawbacks.

For the purpose of this project, each geometric surface needed to be mapped uniquely to a corresponding height map texture. This is because concrete data can be applied (uniquely) to any part of the environment. Due to hardware limitations and texture mapping issues, one texture that covers the entire environment was not deemed feasible, so this was split up into several height maps that each covers a part of the environment. This caused a spike in memory requirements, especially considering the high resolution textures that were needed.

Splitting the height map into many parts also has its share of problems. This technique does not cope well when close to the edges of the height map as the traced ray might sample from outside the texture. This is not a problem for a tileable texture, but this was a luxury that could not be had. To alleviate this problem the best solution in this case was to use mirrored texture sampling although it does not eliminate the problem entirely.

By including lighting information, shadows can be achieved this way as well. For each light, each pixel traces a ray to the light source and checks if it intersects the height map or not. High quality shadows can be produced this way but this would require a huge amount of texture samples, making the program extremely fill-rate intensive, especially for multiple light sources.

One last problem with this technique is the most decisive one. Since it only operates in texture space, the viewing angles to the surface are of great importance. With a perpendicular view angle, hardly any artifacts can be noticed. However, when the angle to the surface becomes too steep, the depth perspective disappears entirely in favor of ugly aliasing issues. The concrete needs to be able to build up on the surface, to potentially large heights, and still look good. But, alas, this cannot be achieved with this technique.

3.5.1.2 Vertex Displacement

The above problems can be averted by opting to do direct vertex displacement. Instead of using texture tricks, the geometry itself is modified. By saving and updating the concrete value per vertex and directly modifying each vertex position the geometry can be shaped in any form. The drawbacks of this technique are for one that the geometric complexity needs to be fairly high to produce good results. Also, since the CPU does the legwork of updating the mesh, synchronization with the GPU is necessary whenever vertices are modified. To keep within acceptable frame rates, there is also the need to carefully way the amount of vertices that are affected. Still, these are drawbacks that can be worked around and this method is in use in the current version of SOE.

3.5.1.3 Displacement Mapping

Another texture based technique that was looked at is displacement mapping. This works similarly to parallax mapping in the sense that it uses height map textures to store the concrete data but instead of tracing a ray per pixel it displaces the vertex in the vertex shader. Since the vertex geometry itself is altered, this means that this technique helps to avoid the most sincere problem with parallax mapping. Although it requires a mesh with a significantly higher vertex count, tests showed that this was not a big concern.

A problem with this technique is that the vertex displacement direction must be specified beforehand. Most often the displacement is done along the normal or some pre-specified axis. Concrete application cannot be easily predicted, since a user can spray concrete where- and however she likes. This poses a considerable problem for this method.

It was later discovered that this problem can be overcome by introducing a second texture that stores displacement directions instead of raw height values. This texture would look and behave similarly to a normal map and be used to push vertices in dynamic directions. The benefit of this is that the geometry itself is never altered in any way. This would allow the program to access the unaltered environment data (that is, before any concrete was applied), something that would potentially be of use for statistical and inspection purposes.

3.5.2 Structure

After the concrete and environment rendering method was chosen a data structure was create the be able to support this. Environment meshes in SOE is constructed out of segments and there are a two reasons for this, concrete application efficiency and rendering efficiency. The bounding volume of a mesh is split into a 3D grid of boxes and each box that contains geometry is called a segment, together the segments make up an environment. Splitting up the geometry like this makes both rendering and updating the geometry a bit more complex but it also provides performance benefits which are needed to keep the simulation running at real time frame rates.

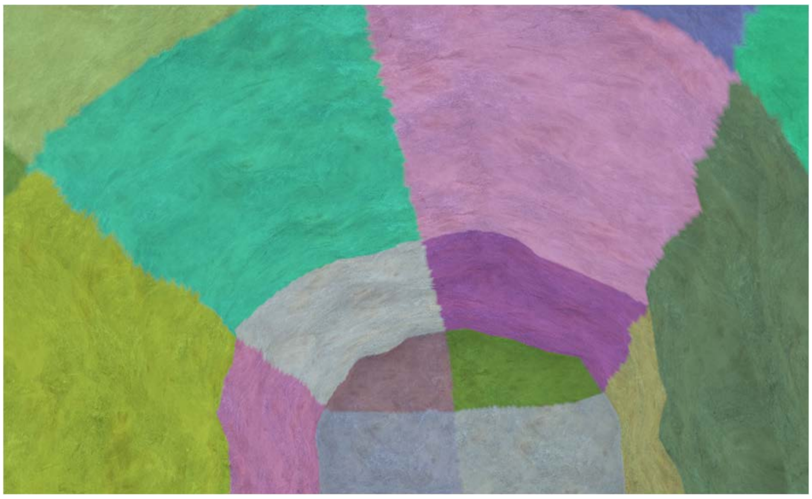


Figure 7: Visualization of environment segmentation.

When both rendering and updates are performed they start by selecting which segments are affected by the current operation. This is done by doing collision tests against the segment bounding boxes when updating and doing view frustum culling against the bounding boxes when rendering. During rendering this applies both for rendering the operator perspective as well as rendering of shadow maps for lights. Because of this, the amount of geometry that needs to be processed during these operations is minimized and a lot of performance is saved.

A segment contains a number of different data collections:

1. The indices of all vertices in the mesh that fall within the bounding box of the segment. This data is used when a segment is updated with concrete. During updates, the software first find which segments that were possibly affected by the sprayed concrete. After that each segment is responsible for updating the vertices assigned to it and uploads the new data to the graphics card. It is not possible for a vertex to reside within more than one segment's bounding box, which means that each segment has a unique set of vertex indices. This ensures that during updates, every vertex is processed exactly once.

2. A set of triangles that lies within the bounding box of the segment. As a triangle is created out of 3 vertices, it is possible that it crosses the border of a segment and so is contained in more than one segment. In this case the triangle is arbitrarily assigned to one of the segments. This triangle data is used to draw each segment during rendering. The unique assignment of triangles ensures that each triangle in the original mesh is rendered a maximum of one time each frame. Besides the first set there is also the possibility to save another extra 2 sets of triangles to render the segment at different levels of detail.

3. A set of triangles that has the possibility to affect the normals and tangents of any vertex assigned to the segment. This extra triangle data is needed during updates when each vertex assigned to the segment updates its normal and tangent after movement incurred by application of concrete. A vertex normal (and tangent) is dependent on all triangles that use it and as a vertex might be used in a triangle that is drawn by a different segment, the triangle set from 2 is not enough. This ensures that all vertices will always have the correct normal and tangent.

3.5.3 Concrete application

Concrete updates are performed in discrete increments and an increment consists of several steps. First, a ray is cast into the environment to check for intersections of surfaces that accept concrete. The intersection test is resolved firstly through a broad phase checking bounding boxes of segments. If a hit is found here, a narrow phase looks closely at the particular segment and determines if and where an exact triangle collision occurred.

When an exact hit has been found, the system looks up all the vertices that could be affected in a certain radius around the hit point. This radius corresponds to the radius of the concrete ray. Concrete adhesion calculations are then performed on each affected vertex. The adhesion calculations return a percentage value from 0 to 100%, where the latter obviously correspond to complete adhesion. Using other data, such as the angle of incoming concrete and the distance to the surface, the exact translation of the vertex is calculated. This calculation is weighted with the adhesion ratio and the 3D mesh is updated. In a linear fashion, the adhesion ratio is also used to determine the amount of rebound.

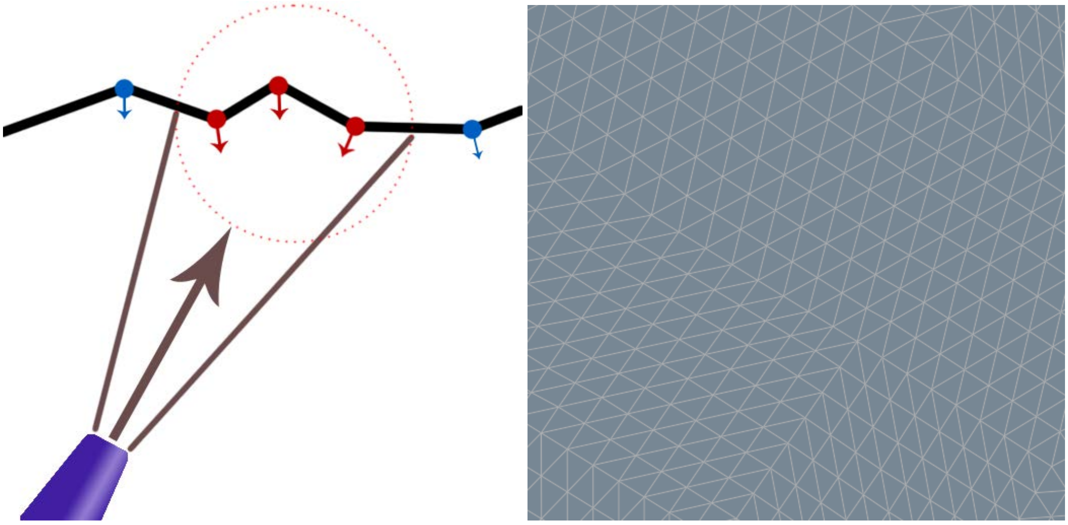


Figure 8: Example of which points are selected for concrete adhesion and how the grid of points on an actual surface might look like.

The adhesion calculation model currently in use is based on (Melbye, 1994), pictured below, and currently takes into account what has been found to be the most influential parameters. The software can be altered to consider additional parameters and the adhesion model can easily be expanded to calculate the results differently.

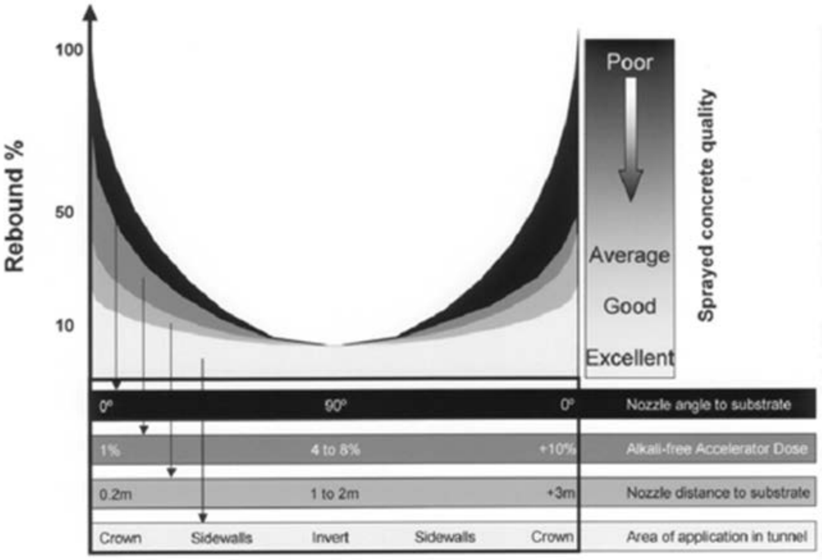


Figure 9: Rebound behavior of shotcrete.

The exact equation that is in use is an approximation of the above diagram and considers distance and angle to the surface. The first part concerns the distance, and is calculated as

$$f = -(d - o)^{C} + 1$$

where **d** is the distance from the surface, **o** is the optimal distance, **C** is an even constant controlling the shape of the curve. Taking angle considerations into account, the actual adhesion can be found by

$$a = f * (N \cdot -I)$$

where **N** is the normal of the surface and **I** is the incident direction. Here, a is defined in the interval [0,1]. This formula represents a reasonable approximation of real world conditions. Presently, it does not take into account surface parameters (such as the type of rock) or accelerator dosage, but can be altered easily.

SOE performs concrete updates at a fixed time interval. This value can be easily adjusted and is currently set at 50 milliseconds, which translates to 20 updates every second. The faster the update interval is the more smooth the application of shotcrete will look but it also means the computer has to do more calculations. A tradeoff is made here where one have to balance the smoothness of the updates against the performance cost of the updates as to keep the target frame rate.

3.5.4 Texturing

The concrete application process causes the environment mesh to change. This is fine by itself, but can cause issues for texturing. Texturing meshes requires texture coordinates, which are pieces of information about how a texture is applied to the mesh. When altering the mesh, especially a great amount (such as creating a large stalactite) the texture coordinates become corrupt. This will manifest itself as stretching the texture which looks far from desirable. Even creating static texture coordinates can be a very difficult task, depending on the complexity of the geometry. Clearly an alternative solution was needed.

These problems are solved by performing so called tri-planar texturing in the fragment shader. This is done by projecting texture coordinates along the X-, Y- and Z-axis and picking the projection that fits best, i.e. the projection axis that is closest to being perpendicular with the triangle surface. When the triangle surface lies on the border between projections a blend is performed between the differing mappings. This system allows for texturing that is always smooth and no effort has to go into generating and updating UV-sets for the geometry. The downside is that a great many texture samples are needed, and hence has performance penalties, but this was decided to be acceptable.

3.5.5 Creation

The features in the simulation and the design of the environment structure put some restriction on the meshes that represent the environment. For example the mesh need to have an even distribution of vertices over the surface area as well as a sufficient resolution of vertices to be able to model the concrete behavior correctly. There are many possible ways to create a mesh that fulfill these requirements.

3.5.5.1 Modeling

One approach is to simply create the desired environment in a modeling program such as Maya or Mudbox. The amount of detail that a simple blasted rock surface requires to appear realistic can be very difficult to create. Creating this amount of detail is both hard and time consuming as well as beyond the modeling skills of any person currently working on the project. Even so, attempts have been made to model this type of environments. Also, modeling tools have been used to enhance or repair an already existing model.

3.5.5.2 Procedural generation

The options of procedurally generating (D. S. Ebert) surface detail on simple meshes can be used to decrease time and modeling skill requirements. Options were explored to create environment meshes from scratch as well as to enhance existing ones using this technique. Procedural generation is quite common in many areas of computer graphics and it is a method of creating geometry or textures from mathematical formulas.

It was found that generating entire environments from scratch is far from trivial realistic results are of the essence. Instead, creating a rough model by hand in a modeling program that has the general shape of the desired environment is a comparatively easy task. Surface detail can then be generated procedurally with the basic mesh as an outline. This method creates a lot better results and has been integrated successfully into the production pipeline.

3.5.5.3 Point cloud data

Point Clouds is a data format where an object or environment is modeled by a large set of points each with a 3D position. These clouds can approximate any object and the more points you have the better the representation becomes. Point Clouds are acquired by scanning an object or environment in reality with a laser scanner. The scanner does this by sending out rays of light which are reflected when they hit an object. The reflected light is measured by the scanner and it creates a point on the position where the light hit the object. Later, a computer can use the point cloud to recreate a model of the scanned object.

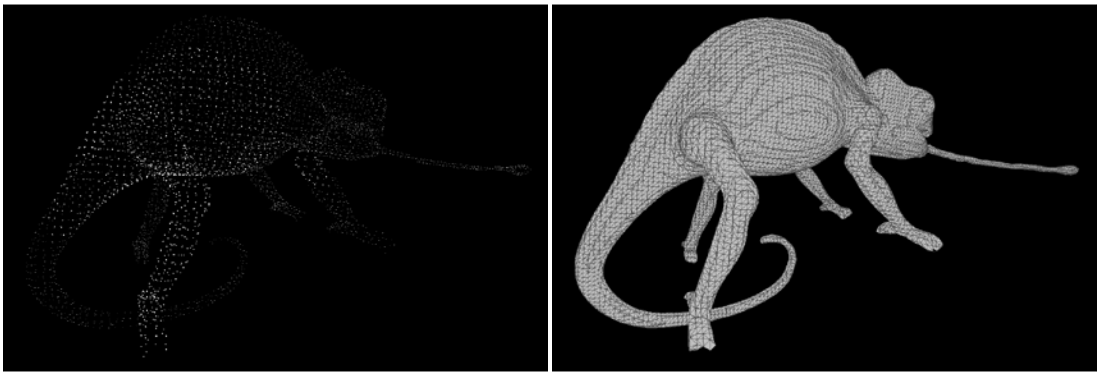


Figure 10: Example of a mesh generated from a point cloud.

This process is quite common today and it is common for construction companies to carry out laser scans of tunnel projects. SOE can obviously benefit from this process and point cloud data models have been acquired with the help of Quanhong Feng at ÅF Infrastructure.

The main problem with this method is to find a suitable algorithm for the reconstruction of the point data into a triangulated model that is usable by the simulation software. This is a fairly well researched area and there are many algorithms that can create triangulated geometry from point clouds. The problem is that not many of the algorithms that have been looked at produce an output that is directly usable by SOE. This is mostly because the need for a watertight mesh as well as a regular distribution of vertices in the resulting mesh. Many algorithms are instead focused on reproducing the geometric object with as few triangles as possible. To automate as much as possible in the environment creation process software has been created that can do most of the work with converting triangle meshes to the appropriate format for the simulation. The process still require some manual intervention at times where reconstruction did not produce the desired result.

3.5.5.4 Environment Mesh Processing

To handle the all the data associated with the environments a few utility programs were created to process the data into the right format as to be usable by SOE.

Maya Plugin

In the case of environments Maya is used in two ways. The first is to import models generated from point clouds and repairing areas of the model that might not have been correctly created during the conversion from point cloud to triangle mesh. The second case is to model a basic environment shape that can be used in the procedural generation in the processing software. When the models are finished a custom plugin extracts the data and exports it to a text based file format. Currently the plugin exports vertices, indices, tangents, and texture coordinates.

Processing software

The environment processing software is a tool written for this project to automate some of the work needed to take a generic mesh and make it ready for use in SOE. The tool takes a raw triangulated mesh as input and produces an output file in the form that the SOE expects. That is, chopped up into segments, and tessellated into a specified resolution. It also has the option of running a procedural displacement algorithm on the model, generating detail variations.

3.6 Graphics

A lot of focus has been put on the graphics of the simulation as this is one of the most important parts to provide the operators with a realistic experience. At the start of the project the software was a very simple prototype that had been developed during the master thesis work. The prototype was based on a rendering and scene hierarchy library called Open Scene Graph, or OSG for short. During the development of the prototype a number of problems both with OSG and the techniques used had been discovered that were considered serious drawback in a continued development of the simulator. At the time OSG was based on the 2.0 version of the OpenGL rendering API. This made it very hard or even impossible to access many of the features introduced in the modern OpenGL 3.0. Also, the scene graph structure that OSG inevitably made use of did not fit well with the team’s goals. Instead, a custom rendering architecture was built directly with OpenGL 3.3 which gave the development team greater flexibility. Creating a rendering system from scratch can be a huge undertaking. On the other hand, the target domain for the simulator was relatively narrow which made this task feasible and successful.

3.6.1 Deferred rendering

When designing the new rendering pipeline for SOE it was decided to base it on a deferred rendering approach instead of the classic Forward rendering paradigm. This is a shift that has become common in the games industry in recent years and it provides a number of benefits as well as some drawbacks.

The major benefit of a deferred approach is that lighting calculations is decoupled from geometry. This works by rendering the scene in different passes. During the first pass, the geometry pass, all geometry is rendered as normal but instead of drawing directly to the back buffer the data is saved in a number of textures. These are commonly called G-Buffers and the data that is saved here is the color of the unlit geometry, depth, normals and the specular properties of the material. This is all the data that is needed to calculate the lighting for each pixel in a later pass.

Table 1. Channel usage in the deferred buffers.

G-Buffer layout			
Normal.x	Normal.y	Normal.z	Linear Depth
Color.x	Color.y	Color.z	Future use
Specular Strength	Specular Power	Future use	Future use

In the second pass all lights are drawn on top of the G-Buffers and each light uses the values in the buffers to calculate their contribution to each pixel. By decoupling the light rendering from the geometry rendering in this way it greatly lowers the complexity of shader development as the shaders drawing the geometry does not have to account for different light setups.

A disadvantage of deferred rendering is that the G-buffers take a lot of memory and a lot of data need to be transferred each frame. It is also much harder to use hardware multisampling for anti-aliasing as multi sampled textures were not supported by the time this was implemented. To solve the problem of aliasing one of the post-process effects implemented was a AA-technique based on edge detection and blurring based on the color and depth values in the G-buffers.

3.6.2 Lighting

There are two types of lighting in the simulation, direct and indirect lighting. Direct lighting comes from spot and point lights and is the types of light that are directly noticeable by a user. They emit light and produce shadows. A point light is a point that emits light uniformly around it, in all directions. A spot light focuses the light through a cone in one direction. These types of lights can either be placed on various spots in the environment or mounted on the robot. All light sources in SOE use the same lighting model which is a fairly simple diffuse term plus a specular reflection term.

Indirect, or ambient lighting, aims to approximate “residual” lighting from the environment and is an important, but hardly noticeable part of everyday life. The ambient lighting is just a fixed value applied to the whole color buffer in the lighting pass. Before ambient lighting is applied a processing pass is performed to calculate ambient occlusion which affects the strength of the ambient value for each pixel. This is done with a technique called Screen Space Ambient Occlusion (SSAO) which uses the depth relation between pixels to approximate how much they are occluded by the rest of the surrounding scene. Ambient occlusion was implemented fairly late in the development of the project but is an important part of the perceived feel. It helps to accentuate small crevices, openings and gaps in the environment which can help the user tremendously in discerning environment details.

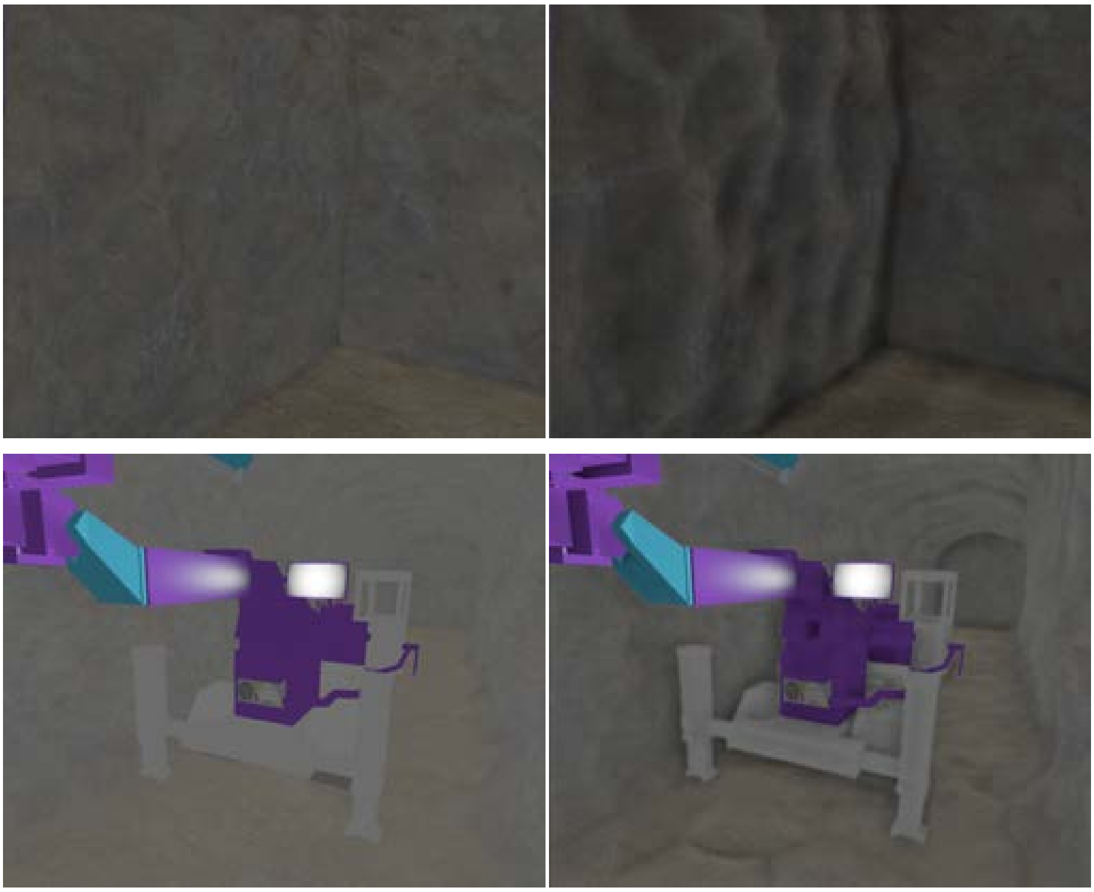


Figure 11: Scenes rendered with(right) and without(left) SSAO.

The lighting situation in a typical tunnel environment usually consists of a small number of very intense lights positioned on and around the robot. This creates a lighting situation with very large differences between unlit and brightly lit surfaces. While the human eye has evolved to be able to handle these situations a computer screen does not have the necessary range to handle this in a pleasing way. To alleviate this problem, a technique called high dynamic range (HDR) lighting is used. Using this, the scene is rendered using a higher range of lighting values and through a process called tone mapping, this high range is mapped to a range that the screen can present.

3.6.3 Shadows

Another subject closely related to lighting is that of shadows. Shadows are very important for human perception because without them it becomes hard to identify spatial relationship between different objects (P. Mamassian, 1998). In SOE, it's very important for users to be able to determine the relationship between the nozzle and the environment.

The shadow algorithm used in SOE is called shadow mapping (Crow, 1977) and is a well researched technique, that is fairly easy to implement and cheap to compute. The shadow mapping algorithm uses a square texture to essentially render an image from the light's point of view. This works very well for spot lights as they emit light in a single direction which can be rendered into one texture. In the case of point lights, since they are essentially spherical, their field of view is infinite and cannot be accurately represented in a single texture (which would require infinite detail). Instead, this is usually approximated by using a number of textures (usually six) from different sides. This process is not hard, but it makes it six times more expensive to calculate the shadow map for a point light than for a spot light. Therefore, point light shadows are not supported in SOE. In practice this is not a problem because in a typical shotcrete environment, the light sources you can expect to find are mostly various forms of spot lights.

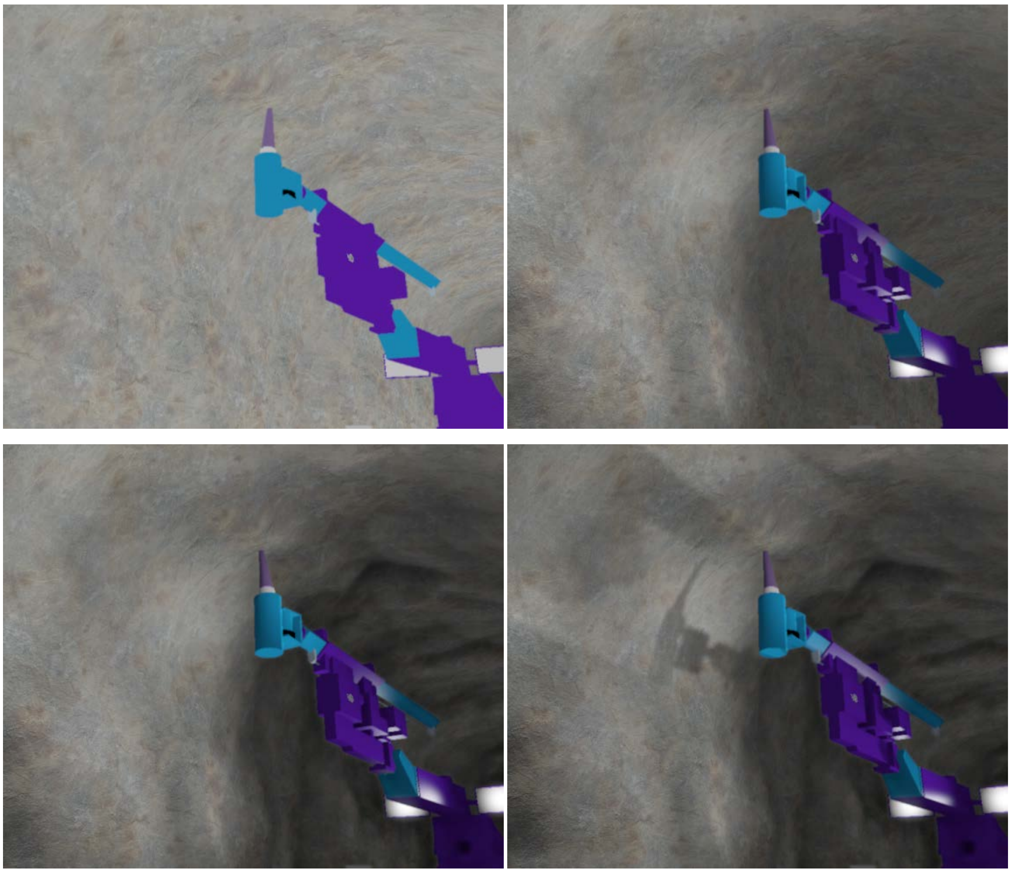


Figure 12: Scene rendered with different effects. From top left to bottom right. 1. Only ambient lighting. 2. Added directional lighting. 3. Added SSAO. 4. Added shadow maps.

The standard shadow mapping solution can (and usually does) make shadows appear jagged and hard edged. The exact shadow mapping technique that is used is called Variance Shadow Mapping (W. Donnely). This technique uses a statistical trick to smooth the edges of the shadow which make them look more realistic.

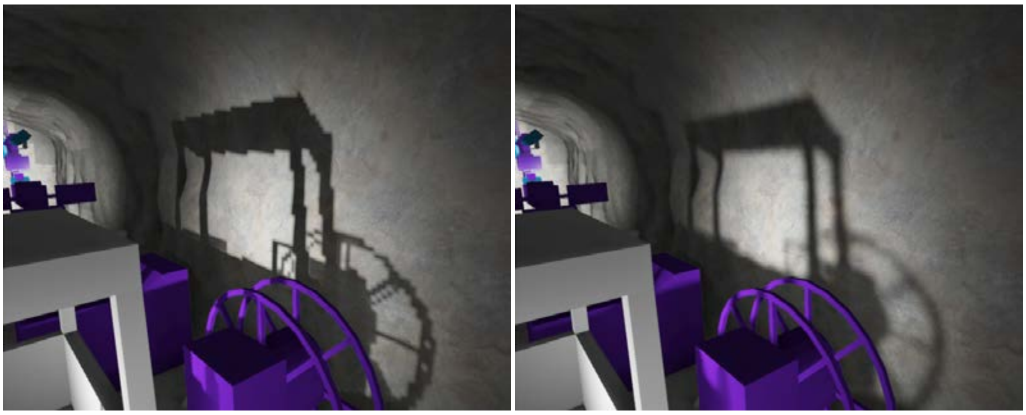


Figure 13: Difference between regular shadow maps and variance shadow maps.

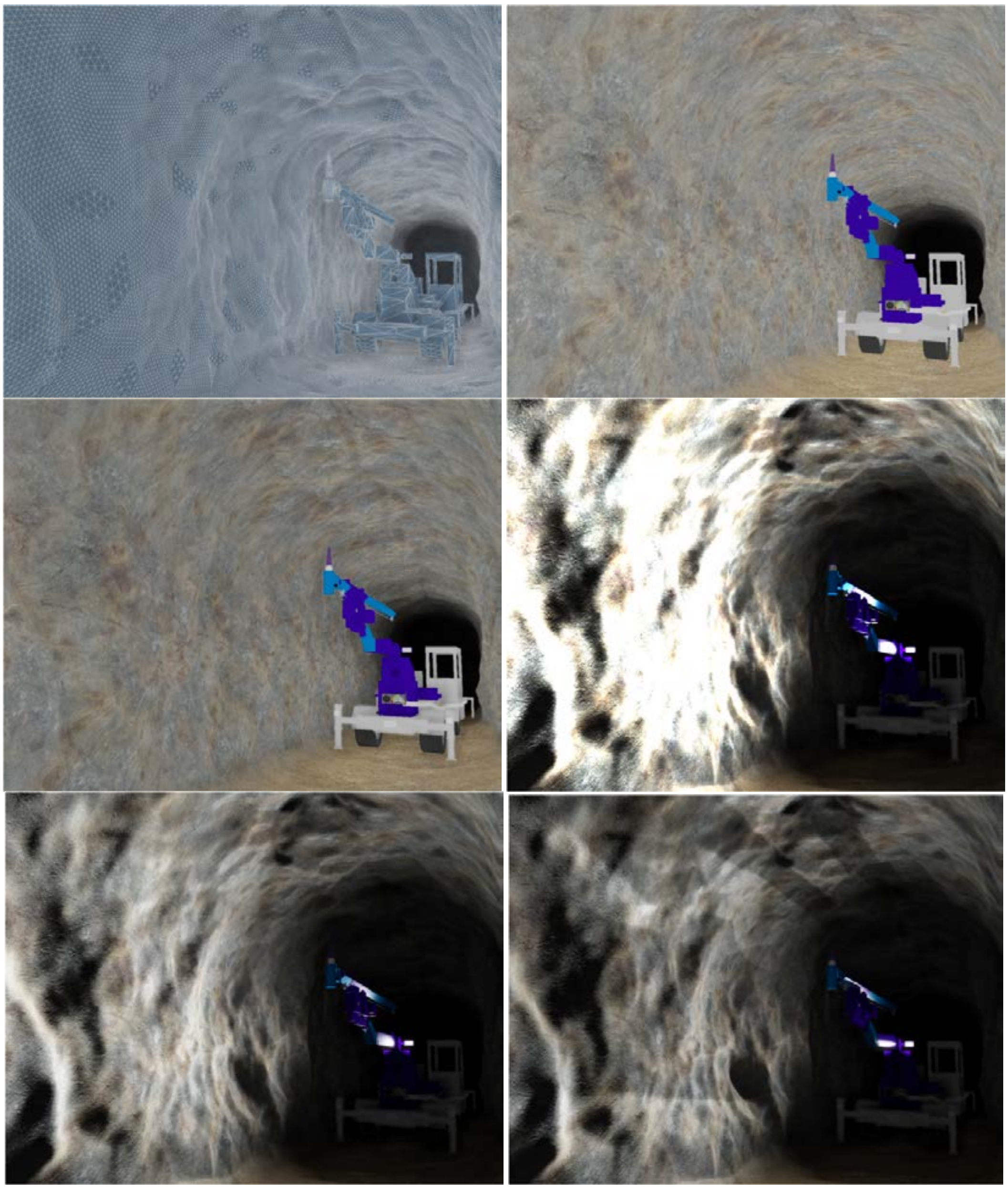


Figure 14: Construction of a scene in the simulation. From top left to bottom right:
1. Visualization of triangle meshes which are the base of the scene.
2. Applied textures.
3. Applied ambient lighting and SSAO.
4. Applied directional and point lights.
5. Applied tone mapping.
6. Applied variance shadow mapping.

3.6.4 Particle Systems

Particle systems is a way to model dynamic systems of particles such as fire, smoke and water in computer graphics. This method is well suited for use in the simulator to represent the concrete flow, rebound, mist and dust that shooting of concrete produce in real life. These effects are important so the operator can know which direction the concrete ray is shooting as well as giving a more realistic environment. The particles in these system does not have any physical effect in the simulation environment but is just used to visualize the actions for the operators benefit.

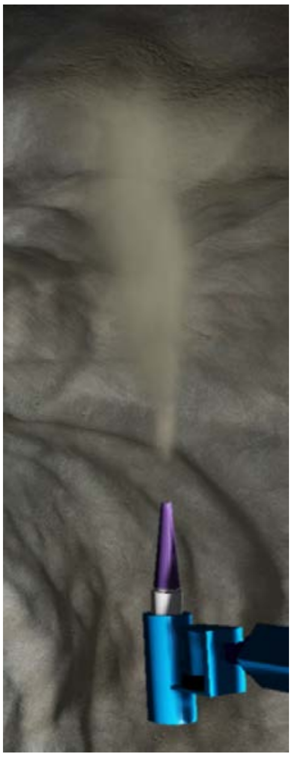


Figure 15: Visualization of shotcrete ray.

3.6.5 Stereoscopic rendering

Stereoscopic rendering is a technique used to create a perception of depth on a two dimensional screen. Ordinary displays can of course show three-dimensional content, but when the image is projected to a screen with only two dimensions it still appears quite flat. Stereoscopic rendering aims to improve this and relies on specialized display devices and glasses to produce a better appearance of true 3D on two dimensional monitors. Examples of this can be found in abundance these days in movie theaters where an increasing number of films can be viewed in 3D.

The technique works by taking advantage of the way which the brain processes input from the eyes. Each eye processes input from two slightly varying viewpoints. These signals are combined by the brain into one seamless image. The separation of our two eyes is what makes humans able to perceive depth instead of a flat image. To display stereoscopic image then, we must render a scene from two slightly different viewpoints and feed each eye the corresponding image. The brain takes care of assembling the images and the outcome is a sensation of depth. Some new pieces of hardware are required to make this work: A high-frequency (“3D ready”) monitor and a pair of 3D glasses.

A regular screen has an update frequency of about 60 Hz. At this frame rate, the brain cannot distinguish between individual frames, and the result is a smooth viewing experience. For stereoscopy, two images need to be displayed in the same amount of time. Since a normal monitor cannot display two images at once, they are displayed in turn and so the monitor needs a refresh rate of at least 120 Hz. Simultaneously, the 3D glasses synchronize with the monitor and feeds each eye the correct image.

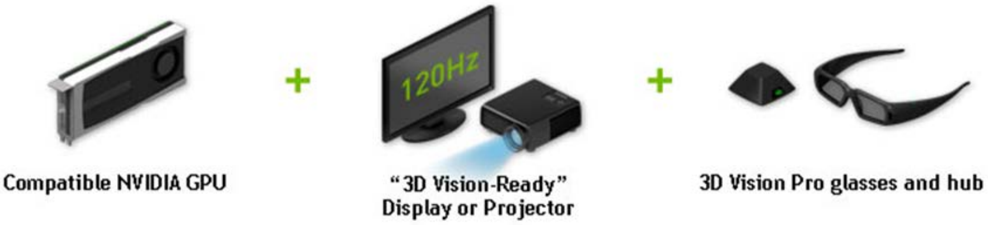

Compatible NVIDIA GPU
+
120Hz
“3D Vision-Ready” Display or Projector
+
3D Vision Pro glasses and hub

Figure 16: Equipment used for quad buffered stereoscopic 3D.

Realism is of high importance in SOE. A higher degree of realism makes the learning process more accurate and pleasant. When talking to test users, they expressed a certain lack of depth awareness which made it difficult to accurately determine distances in the 3D environment. Stereoscopic rendering can help reduce this problem.

Conceptually, the technical aspect of this new type of rendering was not that difficult; set up two virtual cameras instead of one, render the scene from both viewpoints, feed them to the connected 3D-capable screen, and enjoy the experience through the 3D glasses. This puts some strain on the software side which needs to be able to render two images instead of one each cycle, but this problem was not the most important one. In fact, it was hardware limitations that made the process difficult to achieve. As it turned out, even though the current hardware had all the necessary capabilities, the necessary functionality in the low level graphics API was not exposed when using a consumer-grade graphics card, such as the one that was used. To rectify this, a new type of graphics card was purchased that had all necessary facilities. When these hurdles had been overcome, it was a fairly straightforward process to have a stereoscopic version up and running.

Calibration of the three-dimensional effect is of high importance. There are primarily two variables that affect the user. The first one is the distance between the two virtual eyes. The second is the focal plane, the point in front of the eyes where their focus directions meet. These parameters both affect the amount of depth perception as well as the “tolerance” which the user has towards the effect. Every person is different, and when the variables are different from what the user’s brain expects and can handle, tiredness and headaches can occur. It is important that the parameters can be tuned per user and not be too accented.

3.7 User Tests

User tests are important for any software product and SOE is no exception. On several occasions users outside of the development group has been invited to test the simulator. These users have consisted of people from a variety of positions. Arguably though, the most important group of people for this purpose is experienced shotcrete operators.

Two senior shotcrete operators from BESAB have on a number of occasions been invited in to test the product and offer feedback. The feedback these expert users have provided has ranged a number of topics, from graphical adjustments to movement and scale of the robot.

Naturally, user testing has provided invaluable feedback to the development team. If and when the development of the product continues, tests must continue throughout the development process, to iterate on improvements.

4 Result

During the course of the project the goal has been to produce the basis for a software program, usable to educate shotcrete robot operators. At the time of writing of this report there exists a working version of the software produces during this project that accomplish this goal. This section will describe the major features of the simulation software.

4.1 Environment

First of all, the software has the ability to load and visualize environments that a shotcrete robot operator typically works in. Graphically, the gap between the simulation and the real world should not be too great. The environments should also resemble real work environments in size and shape. All this ensures that training in the simulation software will be beneficial. There are a number of ways in the current software to create new environments.

Laser Scanning

Laser scanning is arguably the most accurate way of achieving an accurate representation of a real environment, but it is also the most difficult format to work with. Often the point cloud data, which is the way the data is represented from the scanning process, contains large amounts of jitter and imperfections. It can also be difficult to reconstruct a triangulated mesh from the disjoint points of the cloud.

Manual 3D Modeling

Like any other 3D mesh, it is possible to create an environment by hand in a 3D modeling program, such as Autodesk Maya. The process is similar to sculpting. An artist carves, molds and alters the surface to look like a proper tunnel. However, it has proven very difficult to author environments with surfaces that resemble blasted rock.

Procedural Generation

Procedural generation is the process in which the computer uses mathematical formulas to create content. This process is used to add detail on either laser scanned or hand crafted models.

When an environment have been created, the model need to be processed in a certain way to make it suitable for use by the simulator. To make the creation process as easy as possible a utility program has been written to process the model before it can be loaded into the simulator itself.

4.2 Robot

The simulation software has the ability to load different robots into the virtual environment. The robot currently in use in the simulator is based on the functionality of the Meyco Potenza. Reference photos were used to help the team build a 3D model of the robot. This new robot model has several advantages over the older one from previous versions of the simulator. It has improved mobility in terms of number of joints and axis of operation as well as improved visual appearance. It also conforms well to what operators can expect to work with in the future. While not an exact visual replica of its real world counter-part, it has the same movement options and joint configuration.

The choice of the robot model carries no bias towards any particular brand but was a result of ease of access to reference photos. Both the architecture of the program and the constructed tools have been designed in such a way that adding new robot models, that differ both in terms of aesthetics and functionality, is a smooth process.

4.3 Controls

A simulator of this sort would not be complete without the proper means of controlling the robot. This requirement has been satisfied, and a control device has been acquired and integrated with the simulation environment. The control device itself is an authentic, remote control device of the type that is used in the industry.

The software facilities that have been developed to enable the use of this hardware interface is extensible and can be adapted easily to fit other control devices so that another control device from a different manufacturer can be included.

4.4 Concrete and Adhesion

One of the core aspects of SOE is the ability to actually shoot concrete in the virtual environment. The concrete spraying is accomplished by simulating a flow of concrete from the nozzle of the robot. When the concrete hits the environment an adhesion calculation determines how much of the concrete that sticks to the surface. With this information, the surface geometry can be recalculated. The simulation saves all data associated with the shotcrete spraying for later display. This enables operators to both experience how concrete behaves as well as inspect the result of their work in the simulation.

4.5 Statistics

As it is a simulated virtual environment that is being created, it is possible to not only visualize a real world scenario, but also enhance it in different ways. Depending on the needs of the operator (or the supervisor), different metrics of information can be displayed to aid the learning process. SOE currently records a variety of different metrics.

4.5.1 Concrete Depth Visualization

In the real world, thickness is measured by painstakingly taking manual surface samples on regular intervals, a time consuming and potentially inaccurate task. The virtual world, on the other hand, can make this easier. At each point of the surface, the amount of concrete is recorded. This information can be utilized to display the thickness of concrete. The values are reinterpreted as a color gradient. Blue colors signify low amounts (or no) concrete. The color shifts to green when the thickness is at or close to the optimal thickness (A setting which can be altered easily). Red hues signify a too thick layer of concrete.

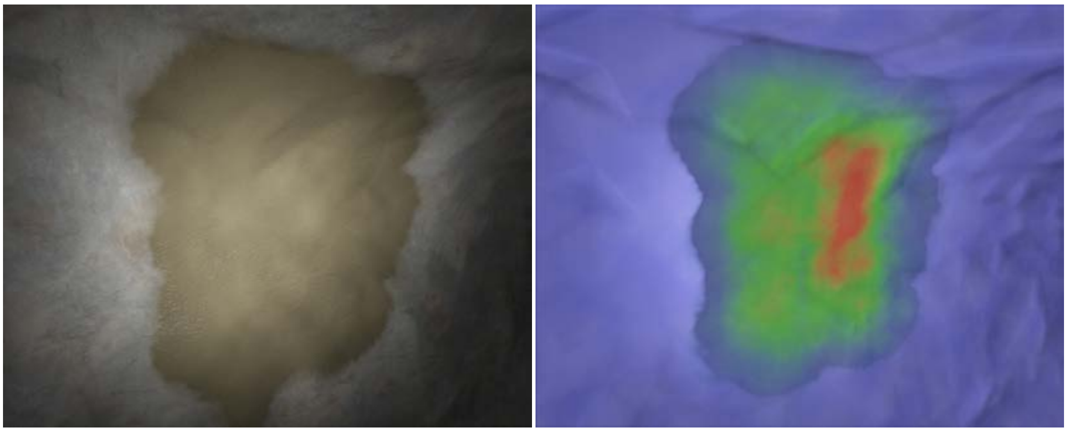


Figure 17: Visualization of concrete depth.

4.5.2 Concrete Usage

Another important metric is to measure the amount of used concrete. The system records information about the total volume as well as the percentage of which actually sticks on the surface. Given these two pieces of information, the waste percentage is calculated. Waste minimization is obviously an essential part of the education process.

4.5.3 Adhesion

The operator maneuvers the machine, positioning the spray nozzle relative to the surface. While doing so, the potential adhesion is calculated. At any time, even before any concrete has been used, the operator can see the effect the current positioning of the nozzle would have on adhesion. During training this data is potentially the most valuable piece of information available to an operator.

4.5.4 Path of operation

In order to achieve optimum conditions, the path followed while spraying plays an important role. Ideally, the nozzle should follow a path back and forth on the surface, gradually rising towards the ceiling. Failing to meet this condition can result in increased waste. This information is also recorded in the SOE and can be displayed as a string of thread, showing how the nozzle moved during the assignment.

It should be noted that any or all of the tools described above can at any time be turned either on or off, depending on the conditions of the current training session. A beginner will likely make use of each of these tools to great extent. An experienced operator, in the other hand, might only sparingly use them to verify the result.

Currently, the exact requirements and needs for the statistics and other feedback during the simulation have not been decided. This area is currently being discussed with experience operators and personnel to get a better view of what is useful and required by the software.

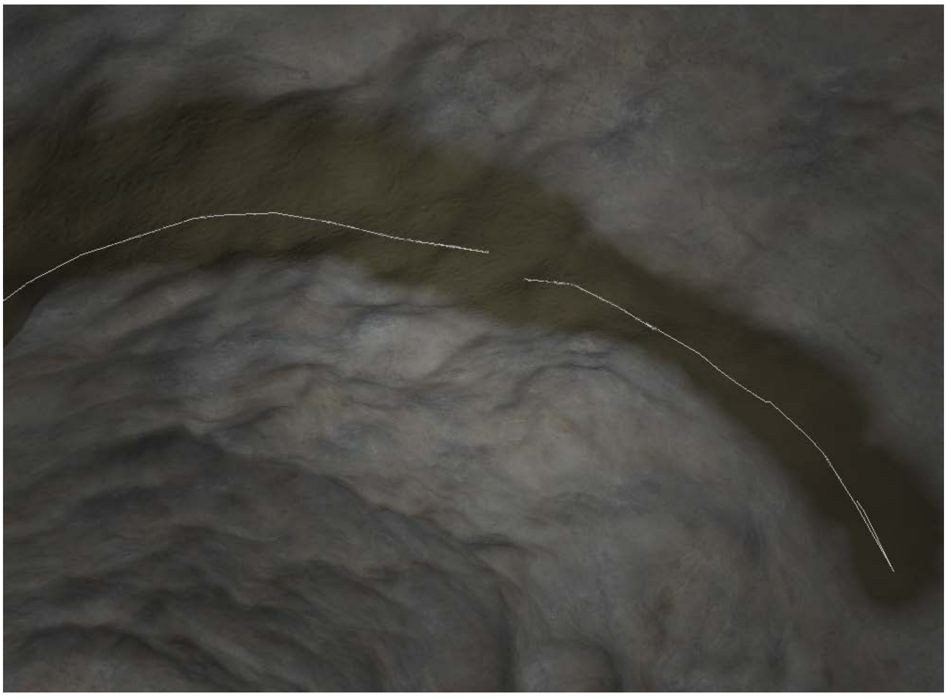


Figure 18: Visualization of path tracking.

4.6 Assignments

To be able to use the simulator in educational programs the system now also supports the creation of assignments. An assignment can specify requirements that the current student need to fulfill and works much like an exam of sorts. These requirements are continuously evaluated during an assignment session and the system can record the result. Requirements can use all of the statistics from the simulation in its evaluation. It could for example specify that the minimum concrete thickness has to be within a certain threshold or that the wasted concrete can't rise above some fixed percent of the total amount used.

Assignments are started from the main menu of the program. A student can choose from a number of assignments (perhaps included in a course). When an assignment has been chosen, the program loads the necessary information and evaluates the assignment in real time, while the operator performs the necessary steps. For example, an assignment can look like this:

Requirement:
- Achieve an adhesion ratio of at least 0.7
Ending Condition:
-Amount of concrete used is 10m3

This program loads the assignment and the student begins. When 10m3 of concrete has been used, the program notifies the user that the assignment is finished. If the adhesion ratio (waste / usage) is higher or equals to 0.7, the assignment is passed. Otherwise it is failed.

4.7 Usability

The virtual SOE camera operates from a first-person perspective. That is, the scene is rendered as if looking through the eyes of a virtual person. We call the virtual embodiment of the user the avatar. Similarly to the real world, the avatar can walk around the virtual scene and look in different directions. The control device used to operate the robot can be used just like it would in a real scenario as it has a direct connection to the virtual robot. The avatar, on the other, hand is more difficult to control. There is no natural way to map the movement of the operator's body to the movement of the avatar. This means that some other mean of controlling the avatar have to be implemented and it has to be easy enough not to hamper the learning experience.

In games, a common solution to this problem is to let the user control the avatar's movement using the keyboard and mouse. Forward, backwards and sideways motions are typically mapped to four buttons on the keyboard and the mouse is used to look around the environment. This control mapping was also the first that was implemented into SOE. As a relatively simple solution, it has the benefit of being widely used and accepted by people familiar with games. However, we cannot assume that the key demographic of SOE are particularly familiar with games or even computers in general. User tests with this solution have shown that this is indeed the case, and the test users were often unwilling to use this control scheme at all.

A partial solution to this issue was to remove the necessity of looking around using the mouse, and only let the user worry about walking. This was achieved by implementing a camera tracking system that automatically targets the interesting parts of the shotcrete process. The interesting part, in this case, is defined to be the nozzle and the concrete's hit position in the environment. As far as possible, the camera in this mode tries to automatically make sure that both the nozzle and the hit point is in frame. This control mechanism functions relatively well and users have been slightly more inclined to move around with this in place.

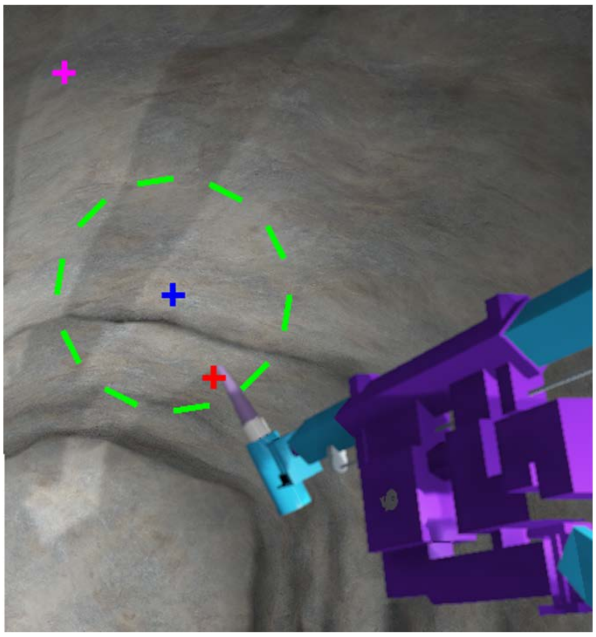


Figure 19: Automatic camera tracking keeps the nozzle(red) and hit point(pink)in view if possible .

Another mode of movement is to have fixed locations and viewing directions for the avatar. This will eliminate the problem of having the user move manually, but it will also impose severe restrictions. This method has been considered as a viable alternative but it has not yet been implemented. If an implementation was to take place, rigorous testing must be undertaken to determine suitable spots and viewing angles which work under all circumstances. It might also increase the workload when constructing new environments if they require specific points the operator can act from as these will probably have to be created manually.

A third solution is a “swap mode” for the robot control. In this case the user can swap between controlling the avatar and the robot with some button on the control device. Having eliminated the need for keyboard and mouse would be an improvement. This scheme would on the other hand require some getting used to. While implemented, user tests have not been performed using this technique.

A solution suggested to us was to let the avatar be controlled in a similar way to that of popular mapping tools (such as Google Maps or Eniro Kartor). The user would then use the mouse to drag the camera, and click on points on the ground to move. This scheme has been implemented but it needs to be evaluated in user tests.

4.8 Hardware and Software Requirements

The SOE is built using modern computational and graphical approaches. As such, the hardware requirements of the program are relatively high, comparable to modern high end games. Mostly, the computational work load is put on the graphics side, meaning that above all, the computer that run it needs to be equipped with a powerful enough graphics card. CPU power is also important, but not to such high regard. The machines that have been used to develop SOE have had the following hardware characteristics.

Processor: Intel Core 2 Quad, running at 2.66 GHz
Graphics: NVIDIA GeForce GTX 275, 2668 MiB memory, 660 MHz
Memory: 4096 MiB
Hard Drive: 1 TB, 7200 RPM
Operating system: Microsoft Windows 7

Note that, in terms of hard drive space and memory, this specification should not be considered the minimum required setup. Hard drive requirements is likely not higher that 4 GiB and memory consumption not greater than 1GiB. As of yet, the minimum hardware specification required to run the program has not been thoroughly tested and is something that needs to be established in the future.

All in all, the program runs effectively on a modern consumer grade PC. Currently, the SOE only supports Microsoft Windows as operating system. It is possible to port the program to other operating systems (such as Linux or Mac OS) but this would likely require a substantial amount of work.

For stereoscopic rendering, a consumer grade graphics card will not work, however. A feature called quad buffering is required for this to work, and none of the major hardware vendors (NVIDIA and AMD) have this option on consumer grade products. This feature is only available on the Quadro series of graphic cards. During development, a NVIDIA Quadro 6000 was used test this feature. It’s comparable to a NVIDIA GeForce GTX 580 in computing power with the quad buffer extension being the only significant difference.

5 Future work

SOE, as it stands today, is still in a very early state that would not be considered commercially acceptable. To get a viable product that can be sold to and used by customers there are more work that need to be done. This section will cover both some points that would probably need to be addressed before the software could be considered complete as well as some ideas for new features and improvements.

5.1 Robots and Controls

One of the main aspects of the simulation is the robots that the operator control. The models that are currently available in the simulation is, as noted above, base on the behavior of real robots, but not exactly the same. To increase the level of realism and real life usability of the product, contact should be made with robot manufacturers to look at the option of importing exact replicas of the real robots that are used in the field today. As the system is constructed with importing new models in mind this process should be fairly pain free if access to 3D models of real robots were available. These aspects are important as there are many different robot designs on the market as well as different designs on the controls that an operator can be expected to use in field work.

5.2 Environments

The tool chain to provide environments from point clouds is not a hundred percent automatic which would be preferable. The problem here is that noise in the points clouds can cause problems for many triangulation algorithms which might need manual work to fix. How good the result will be also depends a lot on what algorithm is used and the choice of algorithm in turn depends on what input is available, i.e. if there are just points in the set or if there are normals for the points as well. There are many algorithms to perform this kind of triangulation and more exploration of this field could possibly find a better fit for this application.

5.3 Concrete parameters

Concrete behavior is one of the pillars of the simulation software and it is important that this appears to be realistic. The current adhesion calculation model depends on very few parameters. Granted, these are the most important ones but there is still room for improvement. For example, most robot models allows the operator to control the concrete volume that the pump outputs. One can also control the air pressure in the nozzle as well as the accelerator dosage. All these variable can affect the optimal distance the nozzle should have from the surface, which in turn affect rebound. If these parameters could be modeled correctly to give an operator a good feel for when and how to use these controls that would provide additional value to the simulation.

Other ideas on this topic is for the simulation to be able to calculate shear strength of the sprayed concrete and have drop-outs occur. Different concrete mixes behave differently and being able to experiment with different volumes of the ingredients and see how this affects the result of the spraying would be very useful. Being able to also model concrete compaction and compressive strength as well as other properties depending on the concrete mix and the operators spraying would exceptional. Worth noting here is that while some of the things here might be possible to accomplish, simulating different concrete mixes and the finished results realistically is most likely way beyond the scope of a real time simulation.

5.4 Education and User interface

The simulator features a system that enables operators to take assignments and automatically get en evaluation of their result. This system still lacks capability to evaluate some aspects of the simulation which will need to be added. Other improvements to this system would for example to be able to set specific areas of the environment that should be part of the assignment evaluation. Also, the user interface and feedback from the system while running an assignment need improvement.

Another important aspect here is to enable the operator to adjust all variables in the simulation that they are able to change in real life. A few examples of this is the speed of different joints on the robot, the air pressure at the nozzle, the concrete volume used per hour and accelerator dosage. All these should be adjustable with the control or through some other interface as they would be changed in real life on the specific robot model that is used at the moment.

6 Conclusion

As described above, the result of this project is a functional software that let shotcrete operators perform shotcrete reinforcement in a virtual environment. User testing and positive feedback from other persons in the industry indicates that it would indeed be a useful tool in the education of shotcrete operators. Even though it might not replace real practice entirely, it is a great improvement as it allows operators to become familiar with the equipment and concrete behavior before being placed in a real situation. The more mistakes an operator makes, the more expensive the work becomes. Using this software, many of the mistakes made by entirely new operators can be avoided or reduced and a lot of money can be saved during their education.

7 Works Cited

Autodesk. (n.d.). *Maya*. Retrieved 11 1, 2011, from http://usa.autodesk.com/maya/

B. Westerdahl, M. J. (2007). *Simulator för träning av robotförare vid sprutbetongsförstärkning, Förstudie.* Göteborg: Visualiseringsstudion Chalmers, SveBeFo Rapport K27.

BESAB. (n.d.). *BESAB*. Retrieved 11 1, 2011, from http://www.besab.com/

Bullet Physics. (n.d.). Retrieved 11 1, 2011, from www.bulletphysics.com

Crow, F. (1977). Shadow algorithms for computer graphics. *Proc. SIGGRAPH*, (pp. 242-248).

D. S. Ebert, F. K. *Texturing and modeling, a procedural approach.*

KranCom. (n.d.). Retrieved 11 1, 2011, from http://www.krancom.se/

Melbye, T. (1994). *Sprayed Concrete for Rock Support.* MBT International Underground Construction Group.

N. Tatarchuk, A. R. (2006). *Practical parallax occlusion mapping with approximate soft shadows for detailed surface rendering.* Association for Computing Machinery.

P. Börjesson, M. T. (2009). *Shotcrete Simulator.* Göteborg: Chalmers University of Technology.

P. Mamassian, D. C. (1998). *The Perception of Cast Shadow.* Trends in Cognitive Science, Elsevier Sciences LTD.

W. Donnely, A. L. *Variance Shadow Maps.* Computer Graphics Lab, School of Computer Science, University of Waterloo.

kaigan tbk
digital thinking

BeFo

Box 5501
SE-114 85 Stockholm

info@befoonline.org • www.befoonline.org
Besöksadress: Storgatan 19

ISSN 1104-1773