



Student: Benjamin Smith
Candidate Number: 83421

Final Report 2006

Degree: Multimedia Digital Systems

Department: Informatics

Tutor: Paul Newbury

Project: 3-D Mountain Formation Simulator

Statement of originality

This report is submitted as part requirement for the degree of Multimedia and Digital Systems at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Signed _____

Acknowledgements

I would like to Acknowledge Adam Kane, lead programmer of Forge effects for his email response, which has helped me to further my research.

Thanks also to Paul Newbury, my project supervisor for his support and calming influence throughout the completion of the project.

Finally I would like to express my thanks to Robert Ayres, Dan Cudlip, Ali Lockie, Emma Avis, and Jemmina Pulsford for their support and motivation, throughout many a late night in the lab, and to Veronique Besnard for her support and sentiments of *“Bon Courage”*.

Summary

This project seeks to push the boundaries of research into the capabilities of merging two leading software technologies; 3D Studio Max and Macromedia Director. 3D Studio Max is an industry leading 3D graphics design package capable of creating complex and scientifically accurate simulations. Macromedia Director MX is a powerful leading platform for the development, design and publication of interactive multimedia. The purpose of this project is to create an interactive 3D Mountain formation simulator and tutorial, by utilising the 3D Graphics potential of 3D Studio Max, and the interactive interface design capabilities of Director, and its programming platform, Lingo. To this end, much of the time and efforts in the successful completion and implementation of the simulator and tutorial have been based on a wealth of research and sequential experiments into the possible methods of execution.

In theory it should be relatively straight forward to create an interface which merges these two software technologies, as their authors have laid down utilities within both programs to enable users to be able to do so. Yet in practise the route to completion has been far from direct, with little previously documented about the methodologies one might use in order to create a bespoke 3D interactive simulator by merging 3D Studio Max and Macromedia Director. This project seeks to add something to this relatively uncharted area, perhaps even to aid the endeavours of future users in the research and completion of their own bespoke packages. In the process of doing this, the user has not been forgotten; the final result is a tutorial which is aesthetically designed, with a wide use of multimedia to aid the interactive learning experience. The tutorial has been based upon the current AQA GCSE specifications for geography students between the ages of 14 too 16, and its primary objectives are to inform and explain the creation of Fold Mountains through the use of multimedia and an interactive 3D simulator.

1 Introduction.....	7
1.1 Project Title: 3-D Mountain Formation Simulator	7
1.2 Problem Area	7
1.3 Professional Considerations	8
2 Requirement Analysis	9
2.2 Weaknesses in Current Teaching Methods	10
2.3 Primary Objectives:.....	10
2.4 Secondary Objectives.....	11
3 Background.....	11
3.1 Interactivity	11
3.2 Interactivities Impact on Education	12
3.3 Review of existing software.....	13
3.3.1 Prentice Hall case study	13
3.3.2 Moorland school case study	14
3.4 Software Choice	15
4 Design.....	16
4.1 Design Principles	17
4.1 Mid-Fi Prototypes	18
5 Implementation	22
5.1 Manipulating Meshes	22
5.1.1 Key Frames	23
5.1.2 Dynamics	23
5.1.3 Havok Physics.....	24
5.1.4 Rigid body objects	24
5.1.5 Bones systems	24
5.1.6 Space Warps.....	25
5.1.7 Shockwave 3D	26
5.1.8 Detailed Picking	27
5.2 Collision Detection	31
5.3 Creating Interactivity	33
5.4 Simulation inputs	35
5.5 Radio button system and plate textures.....	37
5.6 Building a Keyframed animation with lingo.....	38
5.7 Interactivity outside the simulator.....	42
5.8 Picture slide show	44
5.9 Animations	45
5.10 Navigation	46

6 Testing	47
7. Evaluation	47
7.1 Task Specific User Testing	47
7.2 Testing Requirement Analysis	48
7.3 Open ended User study	49
7.4 Targeted User study	50
8. Conclusion.....	51
9. References	53
10. Bibliography.....	54
11. Figures.....	55
12. Appendices.....	56
Front Menu Page	69
Learn Menu Page	69
Fold Mountain Page	71
More about fold Mountains.....	72
Animation Menu Page	79
Animation1 Page.....	79
Animation2 Page.....	79
Animation 3 Page.....	80
Images and other mountains	80
Slide Show	81
Simulation Tutorial	81
Simulation Page	82
Mountain Sculptor Page.....	97

1 Introduction

1.1 Project Title: 3-D Mountain Formation Simulator

The purpose of this report is to document the stages design, implementation and testing of a mountain formation tutorial. This tutorial will utilise the latest technologies in 3-D interactivity and simulations, and will investigate and explore the capabilities of merging the technologies of two leading software applications together (3-D studio max *[1]*, and Macromedia Director *[2]*) in a method not yet tried to its potential.

The motivation for undertaking this project is a combination of my fascination with world geography and mountainous environments, and my interest in the concept of interactive computer simulations. At present there is no such program that utilises the interactive characteristics of a simulation for the purposes of E-learning in this area. This tutorial will be designed specifically to correspond to the curriculum of a GCSE geography student.

“A computer simulation or a computer model is a computer program which attempts to simulate an abstract model of a particular system.” [3]

Computer simulations are fast becoming a valuable tool for modelling natural systems in physics, chemistry and biology, and of engineering new technology, in order to achieve a deeper analysis into the operation of these systems. Traditionally, the formal modelling of systems has been via a mathematical model, which attempts to find analytical solutions to problems. These solutions enable the prediction of the behaviour of the system from a set of parameters and initial conditions.

The aim of this project is to utilise simulation methods to create a tutorial program that works under a smooth user-friendly environment, which enables the user to learn, understand and create their own virtual mountain range. It will demonstrate to them the various factors, the impact of changing these factors, and relate this back to “real world” plate tectonics. There is a lack of such software in the market today. Out of the few existing mountain formation simulators that have been discovered through extensive research, none are sympathetic to the average computer user. In the field of education, this absence is even more noticeable. Thus this program is not intended to perfectly forecast scientifically accurate simulations. The Purpose of this software is to teach and inform in an interactive environment. It will be aimed specifically at GCSE students, and will teach them what is required by the national curriculum.

1.2 Problem Area

There has been a continuing downward trend in entries for geography at both GCSE and A-level since 1995.

“Therte number of GCSE geography entries has fallen by over 20% and a similar decline has been experieenced at A-level since 1998” [4]

There has been little research on the reasons for the decline in numbers but according to a report by the *Qualifications and Curriculum Authority 2005 [5]*, suggested reasons include:

- Pupils’ perceptions of the subject and the way it is taught.
- Structural changes to the whole curriculum.

Thus, a factor in this decrease is the method in which geography is currently taught. It is a dynamic subject constantly re-evaluating its foundations, and re-establishing itself as research is developed in a particular field. However, the teaching of this subject does not reflect its dynamic and interactive nature. Utilising the interactive fundamentals within dynamic simulations will add an extra valuable dimension to the learning process. It would not be credible to expect to create a fully interactive high standard tutorial to encompass the whole of the GCSE subject of geography within the scope of this project. Instead this report will document the design, fabrication and evaluation process for a very specific subject within the wider boundaries of geography, that subject being *the formation of Fold Mountains*. This not only brings a new aspect to the current teaching methods of GCSE geography, but to the realms of multimedia interactive learning. Addressing an area which up until now has been relatively un-trodden by combining two technologies. 3-D graphic animation and interactive learning are joined to create a 3-Dimensional fully interactive simulator, to be used not for analysing or predicting specific scenarios but for the purpose of etching and instruction.

1.3 Professional Considerations

The ethical standards governing the computing profession in Britain are defined by the *Code of Conduct and Code of Practice*, published by the British Computer Society. In conducting this project I am aware that these guidelines must be read, understood and applied to the work and research that I do. The statement below extracted from the *BSC Code of conduct* is the most appropriate to my work.

“

14. You shall seek to upgrade your professional knowledge and skill, and shall maintain awareness of technological developments, procedures and standards which are relevant to your field, and encourage your subordinates to do likewise.

”

[6]

In conducting this project, new and existing technologies will be investigated and used in conjunction with each other, in situations not tested before. I will document the results of all of these experiments, in which technologies and concepts are merged together.

The statements in figure [1.2] are extracts from the British computer society’s code of Practise, and are relevant to my project development. As an active developer looking to research further into the field of

interactive simulation design, I have exchanged ideas with professional developers via emails, and web forums, and sought advice in areas I feel I must research.

- “
- Keep in close touch with and contribute to current developments in the specialism, particularly within the organisation and your own industry.
 - Maintain your knowledge of your specialism at the highest level by, for example, reading relevant literature, attending conferences and seminars, meeting and maintaining contact with other leading practitioners and through taking an active part in appropriate learned, professional and trade bodies
 - Understand the boundaries of your specialist knowledge; admit when you may be required to cross this boundary and seek advice from colleagues with the necessary expertise; do not make misleading claims about your expertise.
- ”

[7]

One such email is shown in the appendix number 3. This was sent to the developer of the *Prentice Hall* case study (shown in section 3.3). The response led to some useful expert resources, which are utilised for the final implementation. Furthermore an ongoing communication with this developer has now started.

2 Requirement Analysis

The National Curriculum requires that students should be given opportunities to apply and develop their ICT (Information Communication and Technology) capacity in all aspects of geography through the use of ICT tools to support their learning.

“In each specification candidates will be required to make effective use of ICT in ways appropriate to the needs of the subject.” [8]

The aims set out below describe the educational purpose of following a course based on AQA (Assessment and Qualifications Alliance) GCSE in Geography, This specification offers opportunities for candidates to, among other things;

‘Appreciate that the study of geography is dynamic, because new ideas and methods lead to new interpretations’ [9]

The idea that the study of geography is dynamic is founded by the very nature of a dynamically changing world. Geography is an incredibly broad subject encompassing social, physical, environmental, and economical aspects of what is happening in the world today. Each of these aspects is interlinked

with each other on an ever-changing course of development. It should therefore, stand to reason that the nature in which geography is taught should reflect the dynamic nature of its content.

2.2 Weaknesses in Current Teaching Methods

A report from the Chief Inspector of schools on teaching of Geography (OFSTED 1995) based on a large sample of English schools paints a bleak picture. It points out that the quality of teaching was good or better than that of the previous OFSTED report 4 years earlier in only 42 percent of lessons. Some of the weaknesses detected are fundamental: challenge, pace and motivation in lessons were often unsatisfactory, weak subject knowledge, “*over-reliance on text books, undemanding activities and insufficient attention to real places and particular circumstances.*” [10]

The process of Fold Mountains is currently included in the AQA specification within section 9.1 shown in **figure 2.4** (Tectonic activity). Where it states:

	Specification detail	Guidance
The Earth’s crust is unstable and creates hazards.	<p>Global distribution of continental plates.</p> <p>Tensional and compressional margins.</p> <p>Characteristic features and formation of fold mountains, earthquakes (focus, epicentre) and volcanoes (composite and shield volcanoes).</p> <p>Occurrence and measurement of earthquakes.</p>	<p>The processes of plate movements should be understood and their role in the formation of fold mountains, earthquakes and volcanoes.</p> <p>The link between earthquakes and plate boundaries to be understood.</p>

Figure 2.1 Tectonic activity within education [10]

So it is apparent that the nature of physical geography is recognised as dynamic and unstable, yet the current teaching methods for this subject do not accord with its dynamic characteristics, with the use of “cold” text books and insufficient attention to particular places and real circumstances.

2.3 Primary Objectives:

- To create a tutorial explaining how Fold Mountains are formed using 3Dimensional simulations, demonstrating the tectonic activities involved and the result of these activities.
- To prompt the user to enter variable values into the program such as: Force(N); Mass; time; rock type.
- To introduce causality where altering any given variable affects the resultant 3D animation.
- To make the program interactive and enjoyable to use.

- To investigate the possibilities of combining 3DS MAX technologies with that of Macromedia Director.
- Program must accord with what is required from National Curriculum.
- The user will be able to control the camera view of the animation as it is in motion.
- The user will be able to zoom in and out of the animation.
- The program will have a tutorial on how to use the simulator.
- The program will also act as a tutorial on other types of tectonic activity.

2.4 Secondary Objectives

- The program will simulate more than one type of tectonic movement.
- The user will be able to ‘play’ with their new mountain range, by creating snow and wind.
- The user will be able to place a character in the scene to explore the terrain.
- The program will have a quiz/game on mountain formations.

3 Background

It is important to establish the extent to which interactivity in the learning environment is effective, and how maximising interactivity will actually bring any gains. This section seeks to define the role of interactivity in education, its effects and evaluate the success of similar interactive tutorials which have sought to achieve learning through simulations. It also justifies the software that will be used to create this program, and brings to attention a primary objective of this project, which is to investigate the possibilities of combining two software technologies that theoretically *should* merge well together in order to create an interactive simulation.

3.1 Interactivity

Interaction has always been thought of as a feedback loop, in a sense a decision cycle. The user will start with a specific goal in mind; and generate a method of achieving this goal, for example by setting variables, clicking buttons, dragging and dropping icons. The plan is then executed by carrying out these actions and finally the result is compared with the original forecast in mind. This process is interactive because the user’s actions dictate the environment, and if well designed this process will be repeated until the user approaches something close to the expected result. The user acts and the environment reacts; the user registers the result, and acts once again. Interactivity is further maximised as the environment is made more responsive to the cognitive needs of the user. This moves more toward the social sense of interaction as shown in figure 3.1.

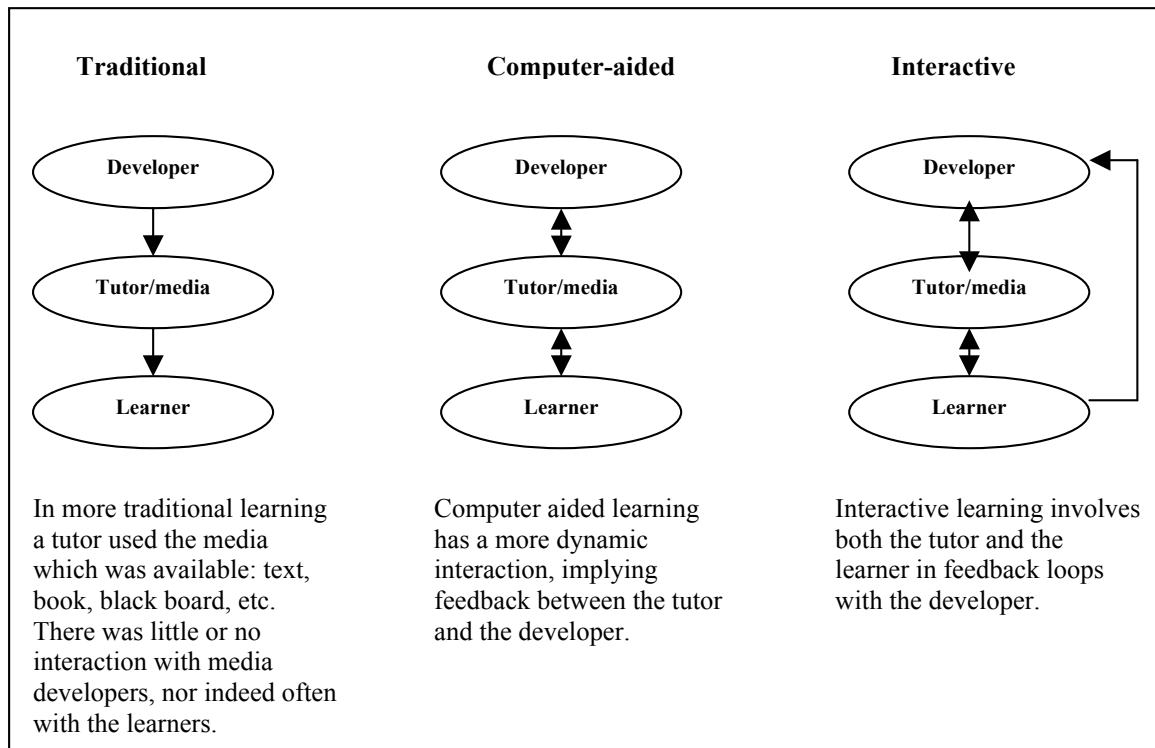


Figure 3.1 Types of Interactivity

3.2 Interactivities Impact on Education

The ready access to full-motion video and sound allows instructors to vastly increase the emotional and motivational appeal of their material, which in turn, should have dramatic effect on the impact that it has on students and their eventual information retention. It is a truism to say that knowledge is not naturally compartmentalised. Many simulation programs raise the issue, because although they are designed with one subject in mind, their use and the learning that can be achieved may cover a number of disciplines. The Mountain formation simulation is designed specifically for the geography GCSE, but other subjects could gain a great use from it. Science of forces and Newton's will be re-enforced; geology and mathematics are also affected.

“Information technology is encouraging this breakdown because it supports the development and use of materials which are naturally interdisciplinary.” [11]

The interruption of information technology in the learning environment, has forced tutors to re-think the way in which they teach. It has been suggested that the technology opens the way to the replacement of teachers by software and training systems. This is not what this application is designed for. With their present power and application, computers and interactive learning provide an extra mechanism in the arsenal of tools available to the learner, and indeed to the teacher, in a sense that it is the teacher who must place it in the appropriate position among all other resources that may be available. The attitude and approach adopted by the teacher will dictate the success of interactive technology in the curriculum.

3.3 Review of existing software

Investigations into the tutorials currently available show that similar interactive simulation approaches have been attempted and to some degree of success. An evaluation of these existing tutorials has been carried out to obtain what might constitute a good tutorial as opposed to a bad one. This looks at the usability of the systems, and examines the technologies that were used to create them. Take sections 3.3.1 as examples of successful interactive tutorials:

3.3.1 Prentice Hall case study

3D Topographic Maps

Developed for Pearson Prentice Hall (a global leader in online learning) as part of the Science Explorer digital curriculum and featured as a Macromedia Showcase Site. This real-time 3D simulation illustrates how topographic maps are created and used to depict changes in elevation. Students are able to sculpt mountains and valleys in real-time and see the changes to the corresponding topographic map.



Figure 3.2 Topographic Map Case Study Prentice Hall

[12]

3D Ocean Waves

Developed for Pearson Prentice Hall as part of the Science Explorer digital curriculum and featured as a Macromedia Showcase Site. This is a real-time 3D wave simulator that demonstrates the connection between wind speed and ocean particle motion depth. The student can modify the properties of an ocean wave and see the effect from any perspective

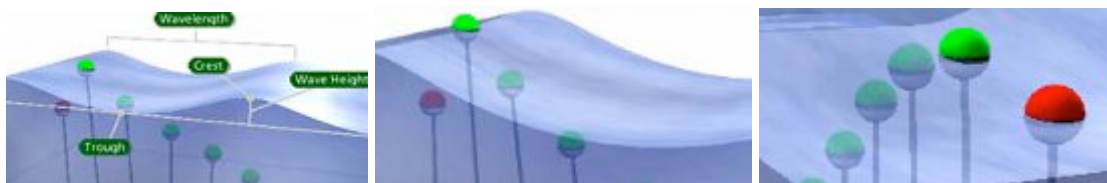


Figure 3.3 Ocean Wave Case Study Prentice Hall

[12]

The appealing factor in both of these interactive tutorial simulations is their aesthetic quality. Their capability to allow the user to rotate camera angles, zoom and pan makes them both very 'usable'. It encourages an experimental environment in which a student may learn. Through the use of the real-time 3D capabilities in Director, these online simulations include water, reflections and mesh deformations and a host of other 3D

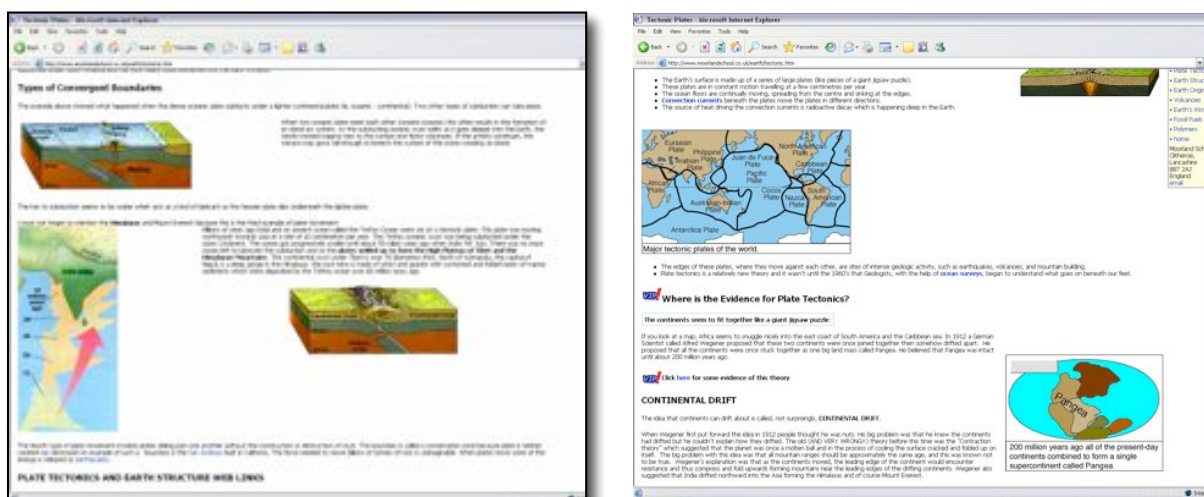
features standards in most cutting-edge 3D tools. The Macromedia W3D exporter allowed ForgeFX's 3D artists and animators to do their art work in 3D Studio Max and export their files into a format that worked with Director. One project requires students to sculpt a mountain in real-time and then be able to see the resultant topographic map.

"Building on top of the foundation that Director MX provides, we were able to create the custom components for this experience." [12]

This simulation is of particular interest and it directly involves dynamically deforming an object in Director. The technologies this tutorial used in order to achieve this however, is not the same as what is required of the 3D Mountain Formation Simulator, as the graphics were created in Director, and not exported from 3D Studio Max, as explained in the email that was received from Adam Kane in appendices 3. The implementation section 5.1.8 investigates the technologies that were used to create this kind of deformable geometry. The examples given above are successful tutorials as they seek to engage the audience in the learning experience, using the concepts of the interactive feed back loops explained in figure 3.1. This is an important point to consider when designing a new tutorial, and something that this project will seek to emulate by implementing a dynamic simulation.

3.3.2 Moorland school case study

Other tutorials are less successful in their goal to engage the audience. Take the example below (figure 3.4), a series of screen shots of an online tutorial about plate tectonics. Instantly it is visible that the layout is unattractive, not seeming to have any consistency between images and text space, and not following any recognisable structure such as Greenberg's grids principle of effective visual communication for GUI design [23]. The result is that the tutorial lacks a sense of horizontal and vertical alignment, making it difficult to locate like components, and creating an unstructured organization. Its contrast is unclear as it does not bring attention to dominant elements, and does not group similar elements by proximity. The consistency within the tutorial is also lacking, with no established location or format for image or textual components. This tutorial engages its students little more than that of a normal textbook. Its only interactivity is the utilisation of Hyperlinks to other related topics.



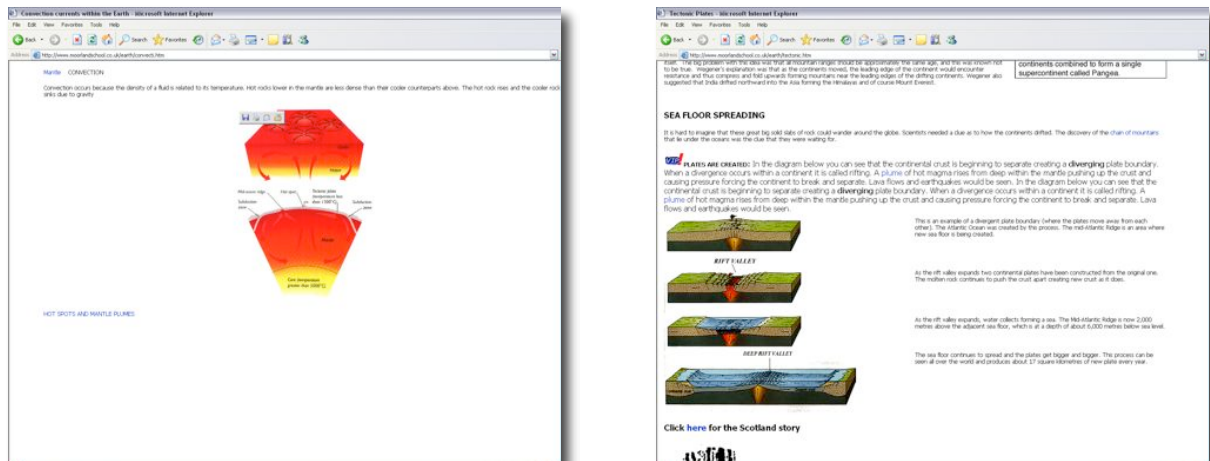


Figure 3.4 Case Study Moorland school

[24]

This tutorial has informed my design, by illustrating the possible misconceptions that can affect the appealing aspect of an interactive tutorial. The inconstancy of the layout and lack of interactivity are aspects that must be improved upon. An efficient use of white space often makes an interface look clean and easy to read. In this example however, it seems awkwardly thought out and is consequently confusing to read. The 3D graphics are a positive point which helps to digest the tectonic movements. This aspect of 3D graphics is something that will implement the final design of the Mountain Formation Tutorial.

3.4 Software Choice

This tutorial will be created using Macromedia Director, and 3D Studio Max. Along side the progression of this tutorial will be an investigation into how these two industry leading softwares can be used in conjunction with each other to create an interactive accurately modelled 3D simulation.

The Havok Xtra within Director provides a physics simulation mechanism for shockwave 3D worlds. It allows setting up collision behaviour, motion, gravity, friction, mass and elasticity of objects, as well as other real world dynamics.

“A Havok cast member is linked to a shockwave 3D cast member. Havok cast members can be created through 3D Studio MAX 4-5-7 (with the Reactor plug-in).” [13]

This should allow accurately rendered graphics to be exported as a W3D file and manipulated within Director. The extent to which these exported graphics can be made deformable within director is thus far less clear.

“The Shockwave 3D exporter does not support soft-body or non-bone-based mesh deformations. You cannot animate the bend of a bird's wing without using a bones system.” [14]

Therefore the creation of this tutorial poses a difficult question: how can one dynamically deform the mesh of a 3D object within Director? There are a variety of ways in which this can be done, each having its merits and disadvantages. During the course of this project, each one of these methods will be investigated and

justified. Thus exploring the functionality of these tools with Director and 3DS MAX is one of the main tasks of this project.

Alternative software such as Java 3D, C++ etc. could be used to create this tutorial. Java 3D is a scene graph based 3D API for the Java platform from Sun Microsystems. It typically runs on top of either OpenGL or Direct3D. C++ is a superset of the C language, an object-oriented programming (OOP) language that is viewed by many as the best language for creating large-scale applications. It was decided not to use these languages to create the tutorial, as this project is as much an investigation into the functionality of 3D Studio Max and Director (lingo), as it is a tutorial. Also to learn and effectively utilise C++ and Java3d would require a longer time-scale. There is much literature on how to interact with pre-rendered 3D models within Director using its inbuilt behaviours, but very little literature illuminating how one might create a more bespoke package which enables manipulation of a model's geometry within Director after it has been built and rendered. For this reason, this project seeks to venture beyond the basic interaction techniques, and actually reveal how one might start to close the gap between a 3D authoring package (3D Studio Max), and a multimedia interaction design package (Director).

4 Design

Before the implementation of the software, design and navigation issues need to be considered. The Initial program design is shown in Figure 4.1 and demonstrates how the various sections of the tutorial interlink with each other. This preliminary design proved to be too narrow, as the nature of the subject which is to be taught far exceeds the boundaries that this design allows. During the implementation of the tutorial it quickly became apparent that a more robust data flow design would be needed, which would be more flexible and enable a wider range of information to be covered.

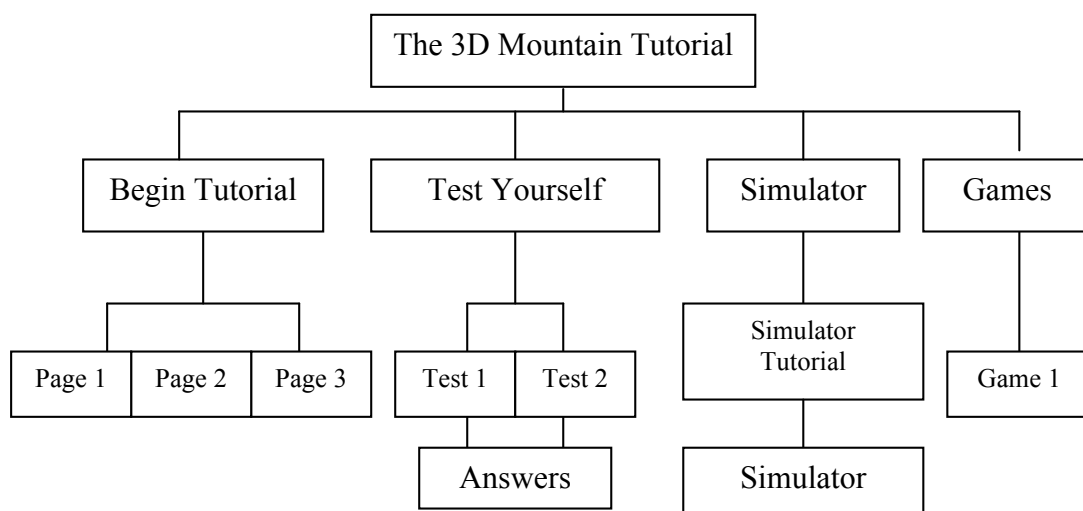


Figure 4.1 Design and navigation concepts

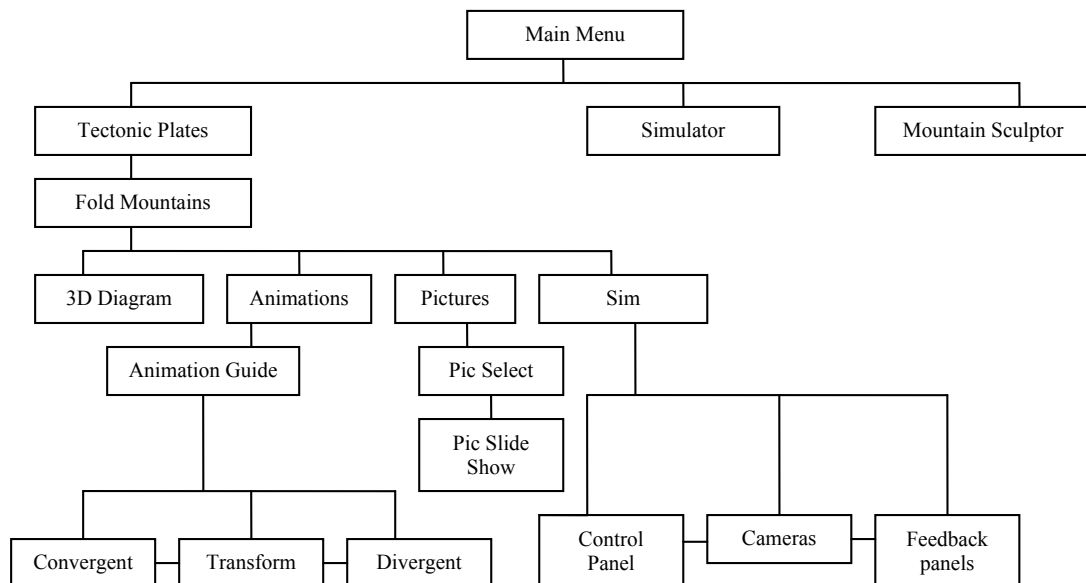


Figure 4.2 Final Design and navigation map

Figure 4.2 demonstrates a more finalised design flow of the tutorial, this implementation allows the inclusion of a much wider range of information and media, such as movies, sound, interactive animation and diagrams, and of course the body of the program, the simulation.

4.1 Design Principles

In order to create a comprehensible and interactive tutorial, the design process has closely followed a series of strict design principles. The principles are explained below along side an explanation of where and how they have been implemented.

1. The principle of metaphor

This involves borrowing behaviours from systems familiar to that which is in the process of design. This is a most important part of the design process as a system which seems familiar will aid the user in understanding how to interact with it. In implementing this principle, metaphors from the real world will be used, such as sliders to set values, virtual tools such as magnifying glasses to control model interactions, and interface metaphors such as a T.V. standby buttons.

2. The principle of feature exposure

To let the user see clearly what functions are available so that a quick visual scan can determine what the program actually does. This will be implemented using various features such as toolbars, menu and Submenu items and dialog boxes.

3. The principle of coherence

This principle will ensure that the behaviour of the program is internally and externally consistent. Internal consistency means that the program's behaviours make "sense" with respect to other parts of the program, such that modifying a variable in one way will hold true for modifying any other variables, creating a set of rules under which the interaction will preside.

4. The principle of focus

Some aspects of a GUI attract attention more than others do. The mouse cursor is probably the most intensely observed object on the screen. Within the program it will be used not only in order to navigate, but as a tool for setting the simulation and manipulating the 3D model. Therefore global state changes will be signalled by changes to the appearance of the cursor, such as different symbols to represent different tools e.g. sculpting, zooming, Camera Panning, and model rotation.

5. The principle of safety

The software should let the user develop confidence by providing a safety net for novices, without slowing down more advanced users. This will be implemented within the program by “are you sure” style dialogue boxes in the most important places.

6. The principle of aesthetics

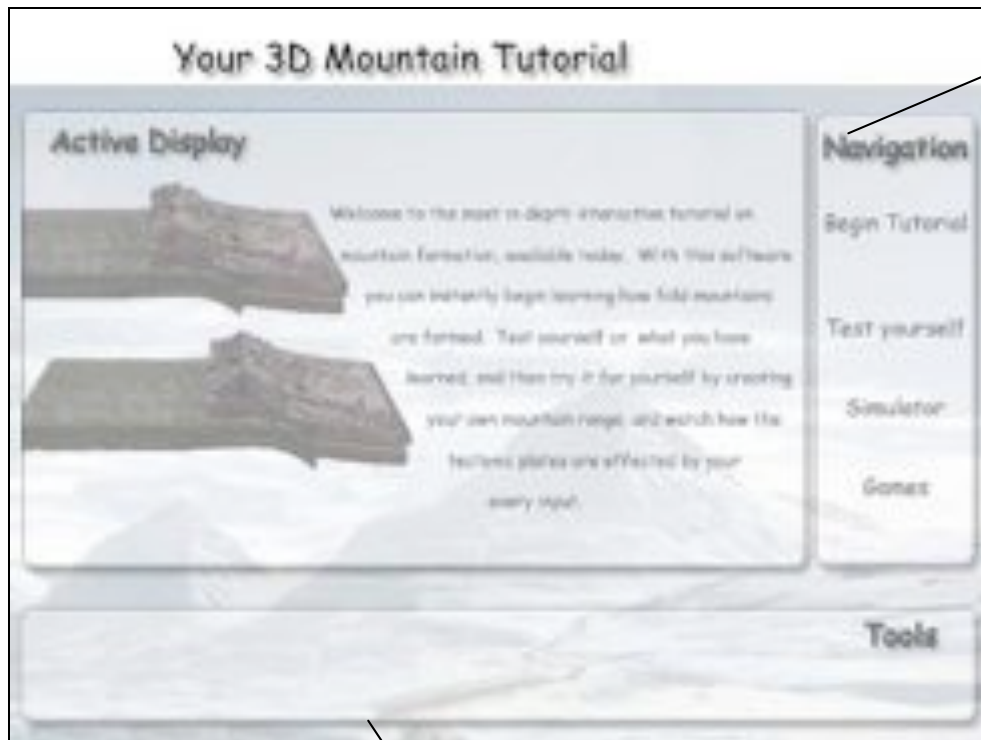
This program explains how mountains are formed, an awe inspiring world landscape, the program should seek to inspire and enthuse its users it must be attractive and aesthetic to use in order to reflect its subject content.

4.1 Mid-Fi Prototypes

The initial program design is based highly on the aesthetic principles explained above, along with feature exposure in the form of navigation menus and tool bars. It is consistent and coherent, using various interface metaphors such as the animation control buttons. The layout is clear and uncluttered, giving maximum focus to the important aspects. Figures 4.2 and 4.3 illustrate how the front-end design of the system might look.

The Active Display panel shows text, information and animations.

This is the Navigation panel with which the user will be able to jump to other parts of the program.



The Tools panel where the user can use various tools to manipulate the simulator, such as zoom and pan.

Figure 4.2 Start page front end design Mid-FI 1

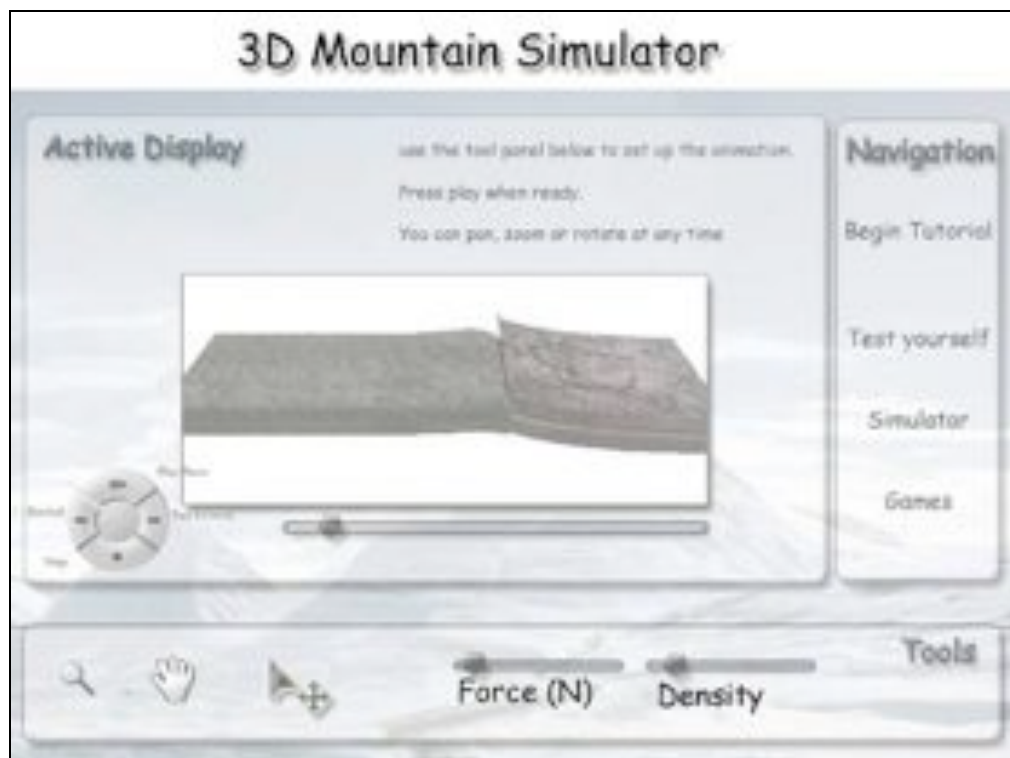


Figure 4.3 Simulation page front end design Mid-FI 1

Figure 4.3 is a possible design choice for the simulation part of the tutorial. Note that the Tools panel is activated with various control mechanisms, compared to the front screen in figure 4.2. The problem with this design is that it is difficult to read, as the colours do not contrast well against each other. It is clear and uncluttered, but is not obvious how to use, and has no instructional documentation. The second prototype below deals with these issues.

The simulation prototype in figures 4.4 to 4.7 has a much clearer contrast making instructions easy to read. It also makes use of some recognition HCI principles, by making the objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another.

Instructions for use of the system will be visible or easily retrievable whenever appropriate, when the user places the mouse over one of the sliders, a user help dialogue will pop up. A unique colour is assigned to each slider to represent different input variables, the feedback panel then displays that value in the same colour as that of the slider, this uses a logical recognition to allow the user to interact without having to take time to understand, how their inputs affect the system.

Outside of the simulation, the design holds consistent, with the navigation menu remaining stable, and the body of text also always in the same panel. All navigations actions are textual based, and all control actions are shown as buttons to reflect real world mapping. It is felt that the second prototype design is more transparent, using clear and concise labelling, following the design principles laid out above.

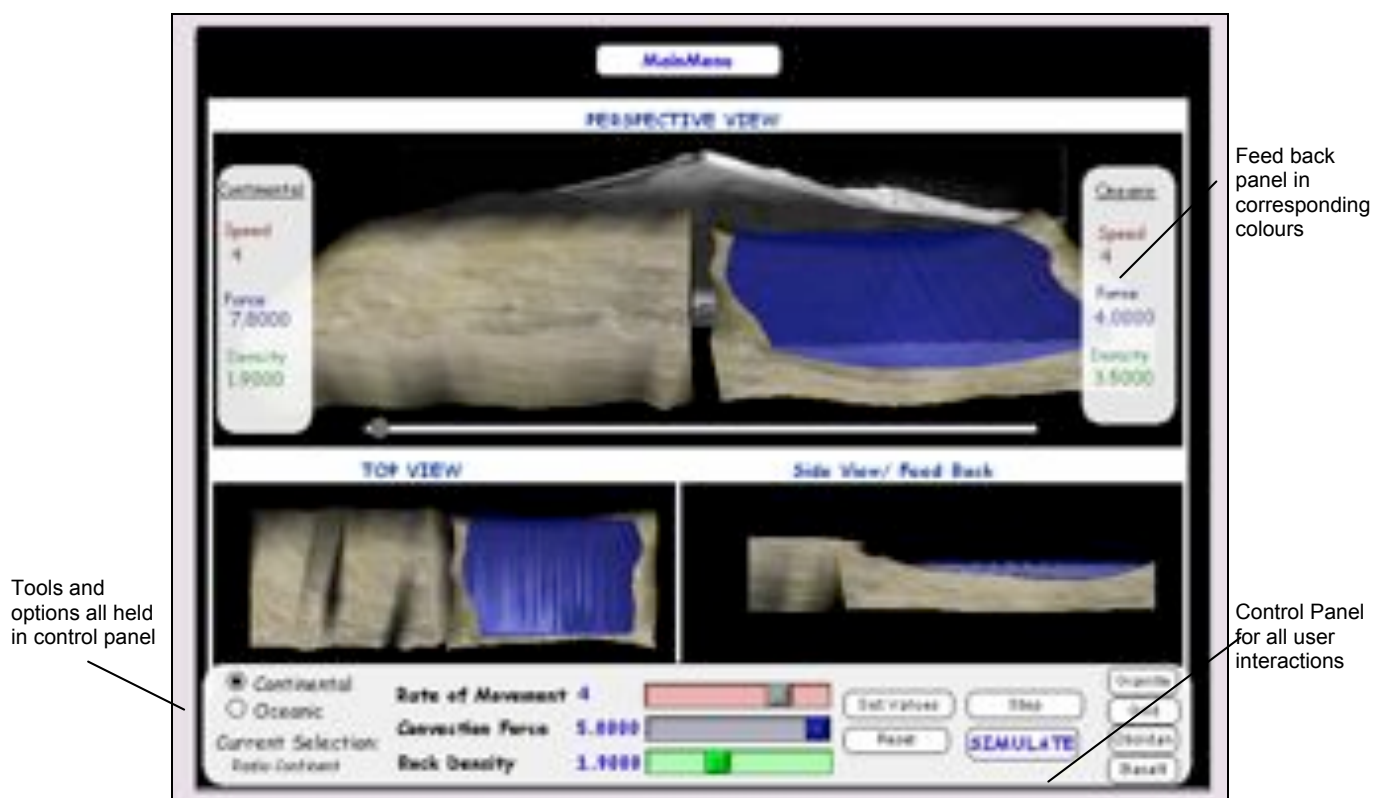


Figure 4.4 Simulation page front end design Mid-FI 2



Figure 4.5 Menu page design Mid-FI 2

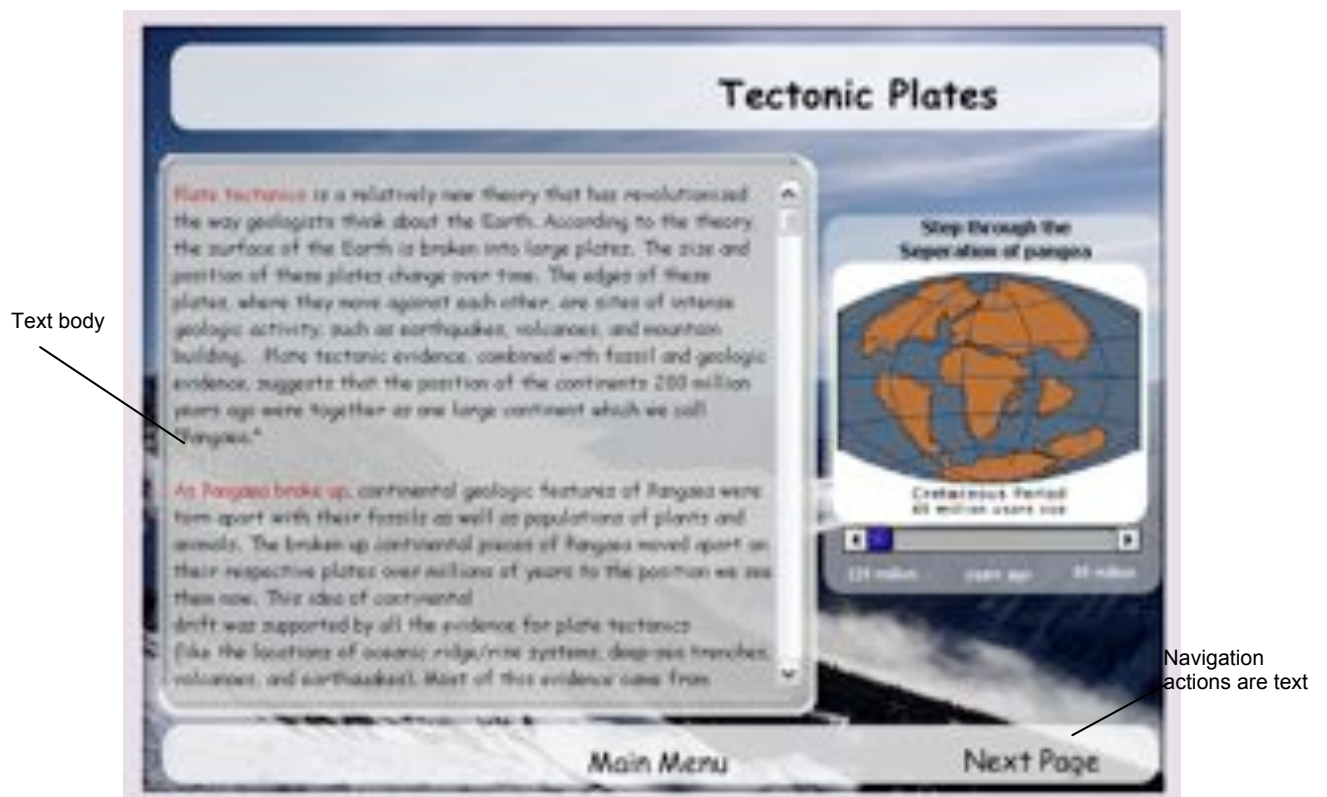


Figure 4.6 Tectonic plates design Mid-FI 2

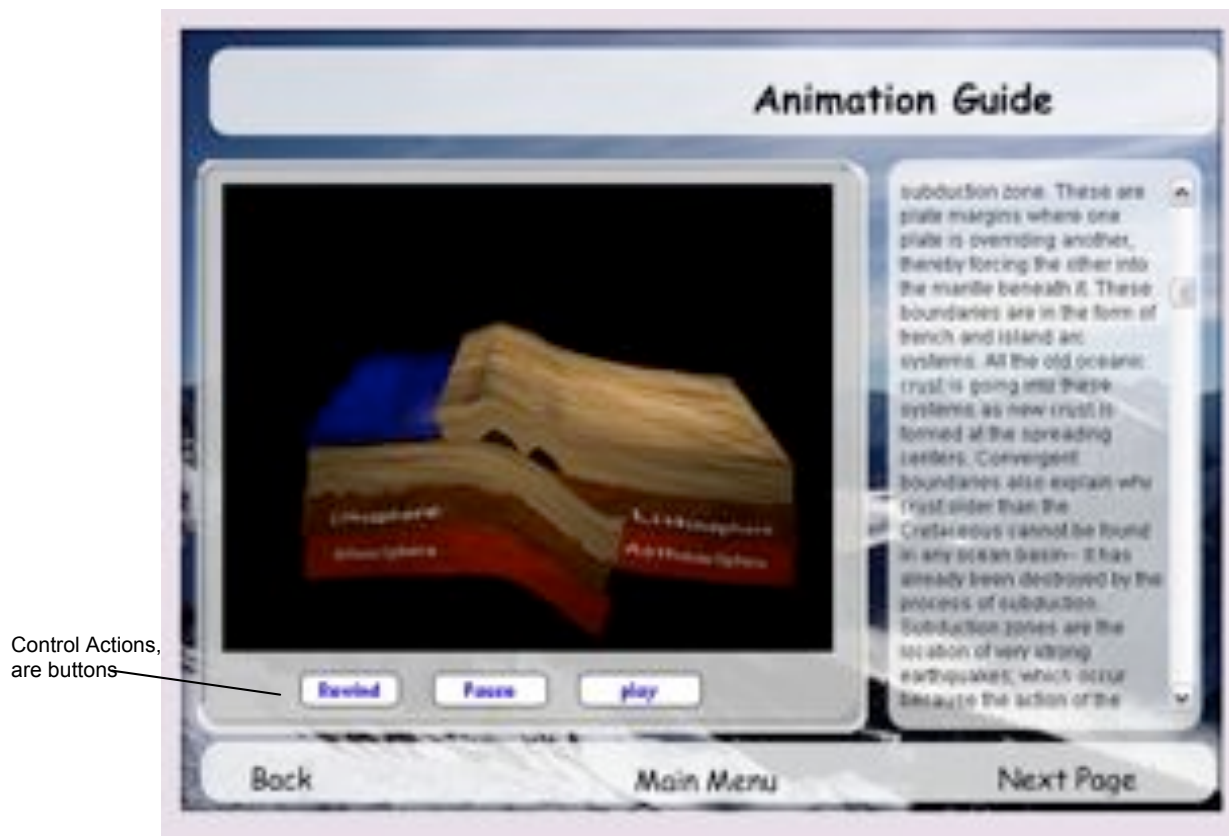


Figure 4.7 Animation design Mid-FI 2

5 Implementation

Throughout The creation of this tutorial, it became more and more apparent that substantial research must be undertaken in order to establish the most efficient way of implementing the simulator, problematic obstacles have been found, and at least in most cases they have been negotiated to a degree of some success. This section identifies these obstacles and their respective solutions, and suggests alternate routes which might have been taken in order to tackle them.

5.1 Manipulating Meshes

A fundamental issue that concerns the implementation of this simulation is how to **dynamically** manipulate the geometry of a 3Dimensional object. By 'dynamically', we refer to the manipulation of the object after it has initially been rendered and in this case, exported from the authoring package in which it was created (3D Studio Max). This proved to be a problem with many inherent obstructions, yet with more than one solution. In order to find a solution which best suited the purposes of this particular software, an investigation was made into merits and limitations of each technology.

5.1.1 Key Frames

Key framing is a term used in animation which describes the process of recording the relative world position of a given object or the vertices which make up the geometry of that object, at a discrete time. Two or more keyframes can be established, usually with the object inhabiting a different set of world space co-ordinates at each key frame. The system can then calculate the world position(s) of the object in-between the points in time specified by its key frames, this process of calculation is called interpolating. Figure 5.1 demonstrates this process.

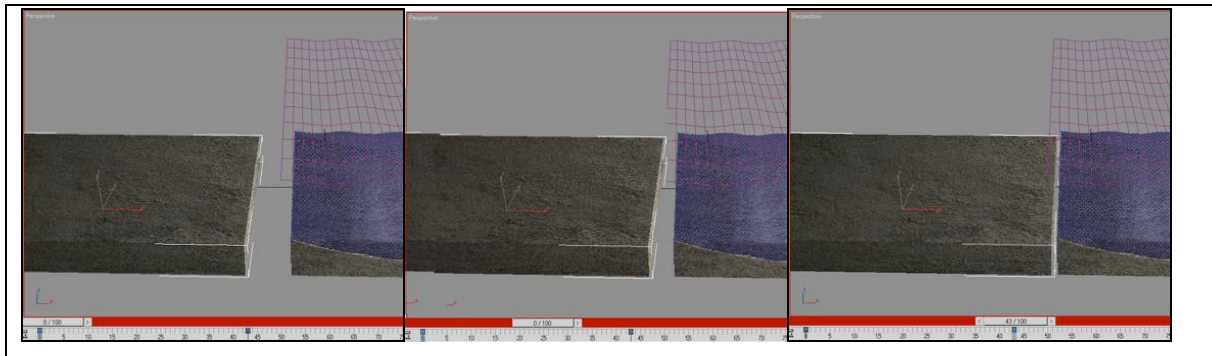


Figure 5.1 Key Frame Interpolation

The time line bar above shows the two key frames that have been inserted; the system interpolates the movement of the object between these two key frames to create the effect of the object moving across the screen.

5.1.2 Dynamics


Initially the first technology that was tried was that which is offered within 3D Studio Max, a utility called Dynamics.

“The term dynamics refers to a system of controls that generate keys to produce animation that simulates real-world physics.” [15]

This Utility enables its users to bypass standard key framing techniques, of tediously assigning individual key frames to smoothly alter the objects geometry, and allows a more scientifically automated approach. The dynamics system enables physical properties to be assigned to each object in the scene (such as friction or gravity), it enables the user to specify which object will collide against another, and then calculates a solution over a range of frames. This produces a very natural result based on surface characteristics and properties. The disadvantages to such a system, is that it produces a series keys which combine together to produce an animation. When this animation is exported, all the physical properties are lost, and all that is left is the key-framed animation. The implications of this are that it cannot be dynamically changed after exportation, unless the key frames can still be manipulated. What is needed is a technology which allows the exportation of objects **and their physical properties**, from the authoring package (3DS Max) into a form that is

recognised by Director, to enable both the objects and their properties to then be manipulated by direct interaction with the user. Such technology does exist, it is called “Havok Physics”.

5.1.3 Havok Physics

“™ is a leading developer of middleware for the video game industry. Used by the world’s most renowned game developers..... Havok’s is the dynamics driving 3ds Max from Autodesk Media and Entertainment.” [16]

Havok’s physics technology relies on a process known as physical simulation in order to provide a dynamic environment for the objects in a scene. This process automatically determines the motion of objects according to their physical properties in much the same way as the dynamics utility will do. The physical simulation splits time into small discrete steps and predicts the motion of each object during each step. Unlike traditional keyframe-based animation, where the animator needs to specify a set of keyframed configurations, physical simulation determines the motion of objects based on their properties. This sounds much like the methods explained above in section 5.1.2; however there is one very important difference. The implementation of the Havok Physics engine allows the dynamic physical properties of the scene to be saved into a format which is recognised by Director. This extension is known as an HKE file. The HKE file contains all information needed to dynamically simulate the same virtual world within Macromedia Director. The Havok Xtra links physical property information (stored within a HKE file) to a display geometry (in a W3D file) via their assigned names. This eventually allows the programmer to create a scientifically accurate simulation that can be initiated in 3D Max, and manipulated in Director. After much experimentation with HKE Files, another problem was soon encountered. HAVOK Xtra only works with ridged body objects.

5.1.4 Rigid body objects

The Havok Xtra assumes that all the objects in the simulated world are perfectly rigid. If all bodies are rigid then it can take advantage of the fact that the geometry of the objects do not vary from step to step so each shape and previous collision can be memorized to speed up the next collision test that is performed. This naturally means that any deformable material cannot be simulated directly with the Havok Extra. Deformable objects can only be simulated to a limited degree using rigid bodies and bone systems.

5.1.5 Bones systems

A Bones system is a jointed, hierarchical linkage of bone objects that can be used to animate other objects or hierarchies. Bones are normally used for animating character models that have a continuous skin mesh. Lingo contains a feature called lingo bones which recognises certain saved bone animations and allows them to be played back, or even mixed together. This means that a physics scene can be generated using the Havok Xtra plug in and manipulated via the bones motions. The draw backs to this are that there is a restriction of motions to that which were exported along with the scene. These motions, although they allow the user to control some aspect of how the geometry of the shape will alter, they do not look as realistic as previously hoped for. An example of the outputs that can be produced using the bones system is shown below in figures

5.2 and 5.3. Director's Bones player manages a queue of motions. The first motion in the play list is the motion that is currently playing or paused. When that motion finishes playing, it's removed from the play list and the next motion begins. Creating new motions can be done by combining existing motions. For example, a colliding motion could be combined with a collapsing motion to produce a subducting tectonic collision.

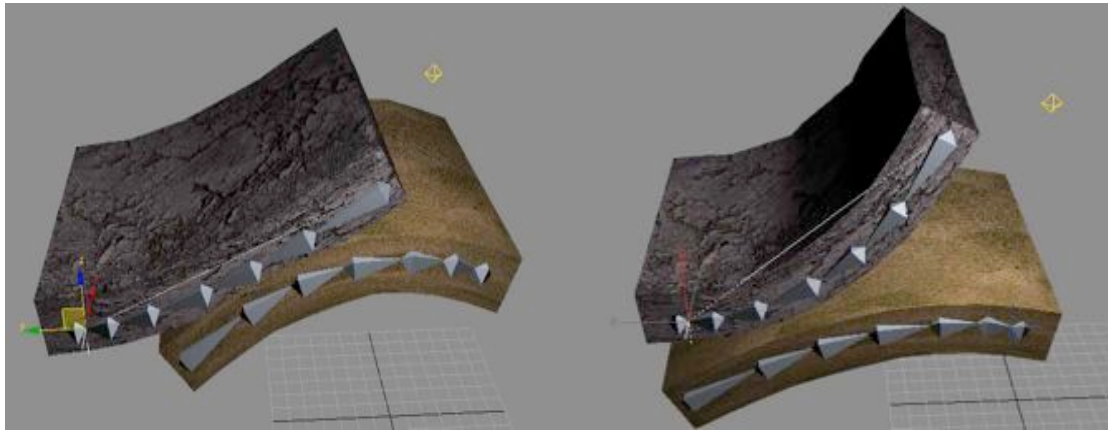


Figure 5.2 Bone animation

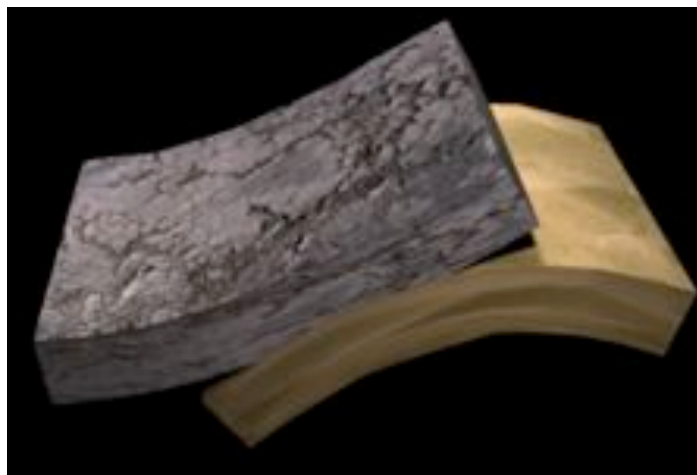


Figure 5.3 Rendered Bone animation

5.1.6 Space Warps

After establishing that the Havok physics engine enables the programmer to define a set of physical properties, but does not work with deformable bodies, and that bone systems are limited to those motions that were set within the 3D authoring package. A conclusion was reached that these methods would not suffice. What was required was a computationally calculated simulation (as opposed to a pre-defined keyframed simulation). Also it must be a simulation which was not limited to a pre-determined number of motions, as defined by the bones system. The investigation moved on to a technology known as Space warps. This enables Havok Physics simulations and also allows direct manipulation of object geometry. Figure 5.4 below, shows two such space warps imposed on the geometry of the scene, (the push and wave warps).

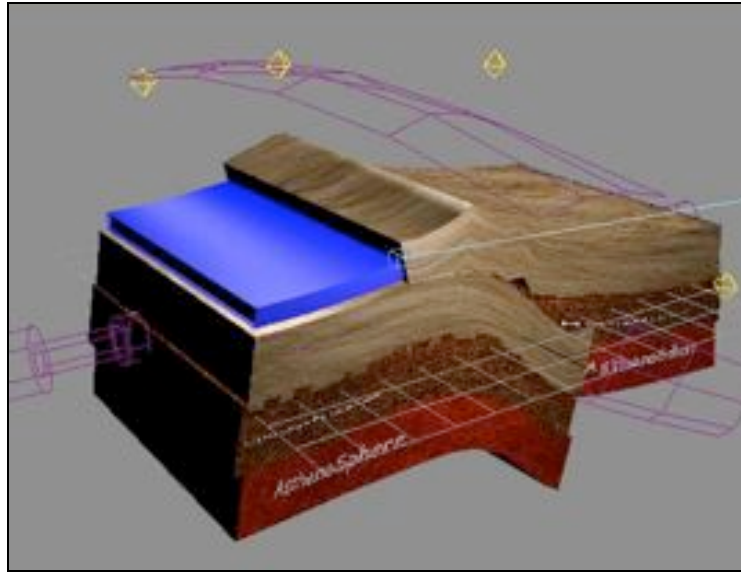


Figure 5.4 Space warps

The push space warp provides a point of force away from the pad of the hydraulic jack icon. A negative force pulls in the opposite direction, in figure 5.4. The push space warp is invoking a force on the oceanic plate, moving it towards the continental and creating a constant pressure. In dynamics, applying a force is the same as pushing something with your finger. The Wave space warp creates a linear wave through world space. The wave warp was used here to create the effect of a subduction of the oceanic tectonic plate. Both space warps are shown in more detail in figure 5.5. At this point the simulation is based on real physics and the geometry of the models can be altered using the space warps, however, a fundamental problem became very quickly apparent. Space warps cannot be exported, upon exportation, they become extinct. This solution therefore cannot be applied to the purposes of this project.

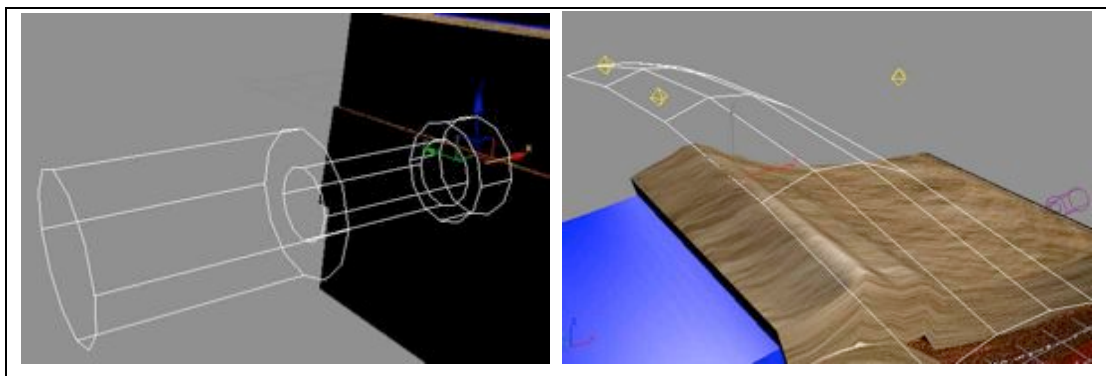


Figure 5.5 Push and Wave Space warp

5.1.7 Shockwave 3D

After research into how to export 3D scenes and animations from 3D max, without losing any of the information which designed the animation, it was found that a file format called Shockwave 3D (W3D), is recognised by Macromedia director and provides a utility to export information such as geometry resources, animations, material resources, texture maps, shaders, lights, bones, and cameras.

Figure 5.6 below shows the analysis of a shockwave 3D file.

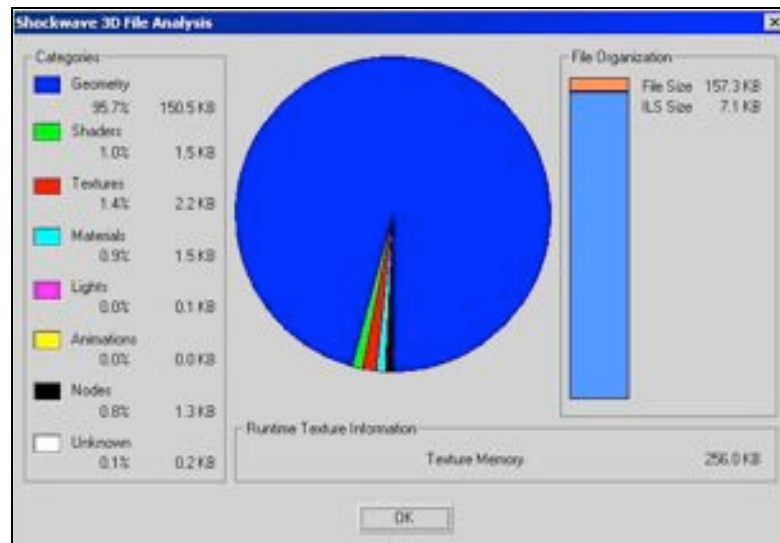


Figure 6.5 Shockwave file analysis

It was decided that the geometry of the objects should be established in the 3D authoring package (3D Max), this geometry could be exported along with all the components which comprise the scene, such as the cameras, and textures etc. The actual simulation can then be performed within Director. This way the simulation utilises the benefits of implementing complex and interesting graphics from MAX, and maintains the interaction through Director. Naturally the next step was to establish how to program the simulation within Director. In order to do this, information and advice was sought from various resources, including web forums, books and even emailing professionals within the industry for advice, one particular technology which continued to be recommended was that of Director's #MeshDeform function. This can be implemented on a 3D model via a programming language called Lingo. The #MeshDeform function provides a low level of control over the geometry of a shape. In short, it allows you to change the positions of vertices and faces of a model in real time. A critical philosophy behind the mesh deform function is that being an abstract tool for controlling geometry, the success of using this function depends heavily upon the organisation of the code. In addition, building the geometry with code within Director will give a critical advantage of understanding of the structure's geometry, this way it becomes more apparent as to how the structure can be changed. This became the approach that was first used. By constructing the models with pure lingo and experimenting with the #MeshDeform modifier, various tests were run to establish how the makeup of a shape can be dynamically altered.

5.1.8 Detailed Picking

A technique known as picking was used to modify the geometry of a plane in real-time. The Plane was created using pure Lingo code within Director, the code used to create this plane can be seen in appendix 65. The purpose of this experiment is to see if when the user clicks on a specific part of the plane, the faces and vertices that make up the area which has been clicked will transform the geometry of the face, creating the

illusion of the area rising. This is illustrated below in figure 5.7. It can be seen that the user has clicked on various parts of the plane, thus altering the relief of the geometry.

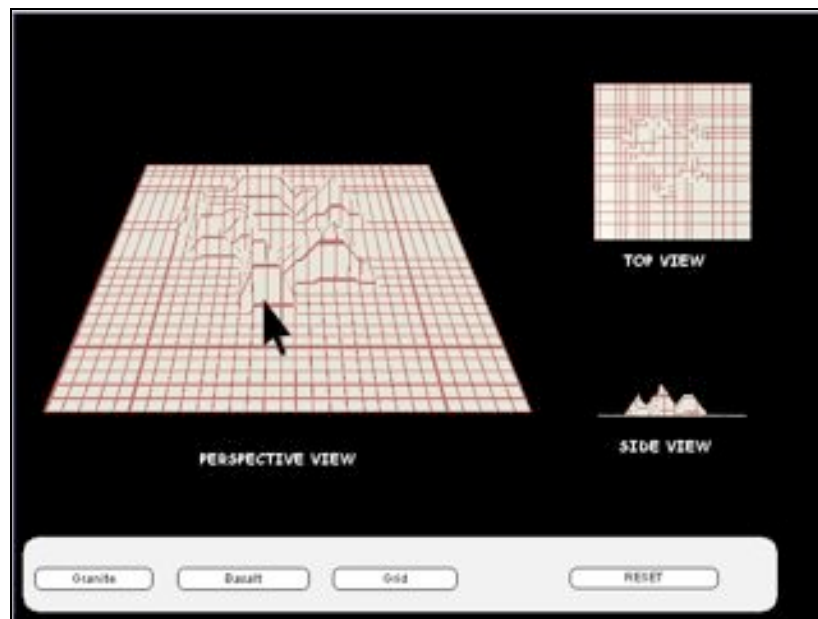


Figure 5.7 Detailed picking of a plane

From this experiment it has been established that the shape of a plane can be altered dynamically, even after the plane has been originally created. The code below in figure 5.8 demonstrates some of the key points in the process of this experiment. Firstly, establishing which point to deform, by using the #clickloc function. Then retrieving the relevant data which defines that point in world space using the #faceID and #vertlist functions, before finally creating a new vector for that face, and reassigning it.

```
global scene

on beginsprite
  origin = point(sprite(spritenum).left, sprite(spritenum).top)

on mousedown
  pt = the clickloc - origin
  -- get detailed info
  modellist = sprite(spritenum).camera.modelsUnderLoc(pt, 1, #detailed
  --make sure that a model has been hit
  if modellist <> [] then
    -- find out which face was hit
    whichface = modellist[1][#faceID]
    -- find out the 3 vertices that face uses
    vertlist = scene.model[1].meshdeform.mesh[2].face[whichface]
```

```

cnt = scene.model[1].meshdeform.mesh.count
meshnum = modellist[1].meshID

--cycle through the 3 vertices
repeat with x = 1 to 3
  --find the exact vertex location
  vert =
scene.model[1].meshdeform.mesh[meshnum].vertexlist[vertlist[x]]
  -- add a small amount to that location
  newvert = vert + vector(0,0,5)
  -- set the vertex to the modified position
  scene.model[1].meshdeform.mesh[meshnum].vertexlist[vertlist[x]] =
newvert
end repeat
end if
end

```

Figure 5.8 Detailed picking Code

In the final simulation, the geometry deformation will not occur when the user clicks on a specific region, but when a collision of two plates is detected within a specific contact point. The natural evolution of this experiment was to see the effects of detailed picking, on an object with 3 dimensions such as a box, as opposed to an object with only two dimensions such as the plane described above. This proved to be a more complex matter. In order to demonstrate why this proved to be more complicated one must first look in more detail at the make up of a 3D object, and the terms used to describe this makeup within Director. An object such as a plane, is made up from a series of faces. Each face is triangular in shape and has a unique index number. Each index number points to a vector containing the three vertex points which hold the positions of that given face in world space. This is shown in figure 5.9 below.

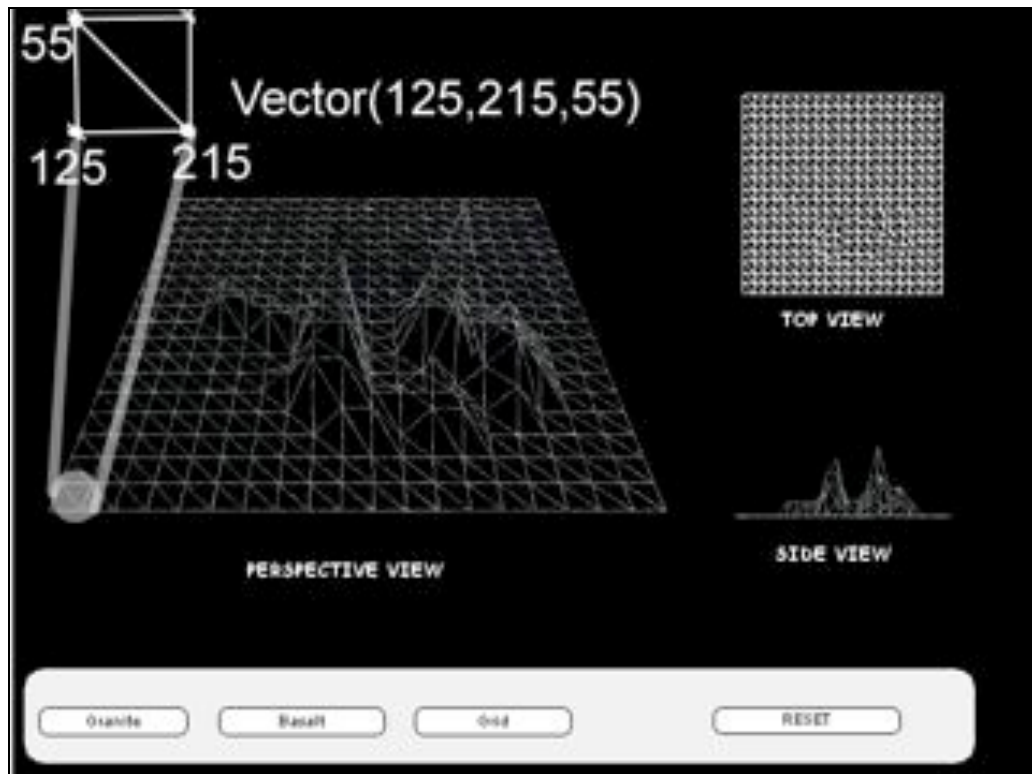


Figure 5.9 Construction of 3D model

Figure 5.9 above shows the wire frame of the 3D model, it is clear that the model is made up of a series of triangles (faces). Each point of a given triangle refers to a vector, which determines its position. Figure 5.10 below demonstrates how this method of vector based faces can then be used to create a 3 Dimensional effect.

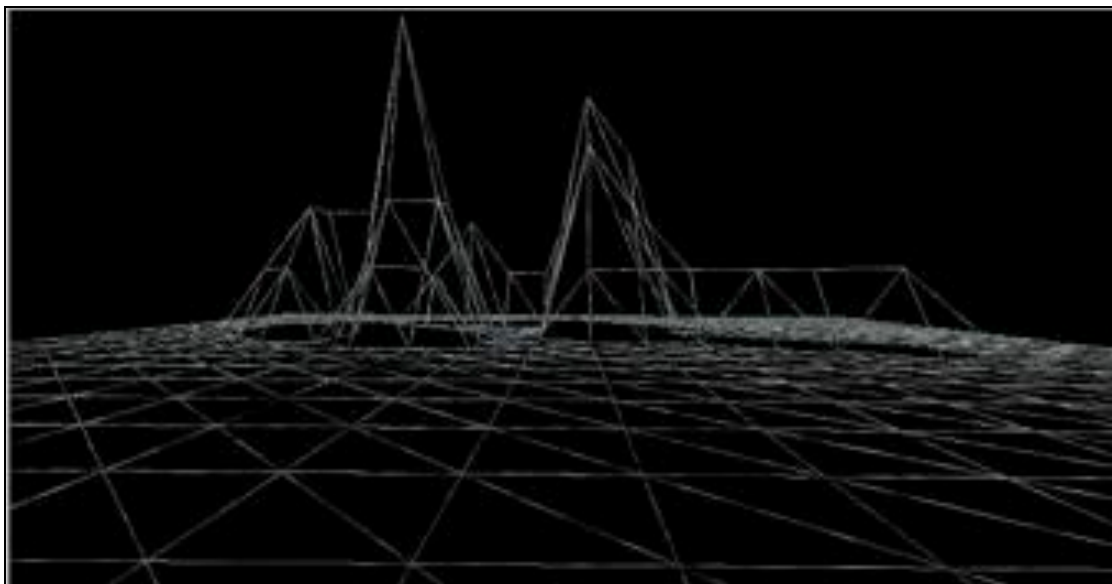


Figure 5.10 3D polygon world

The problem that arose when trying to apply this same picking technique to an object with three dimensions was that a box consists of no less than 6 planes. The issue here is how does one establish which plane's

geometry must be affected? This required some critical thinking and research into how primitives such as boxes are created within Director, as it happens each plane that constitutes the box, has its own unique index. Therefore when the mouse is clicked upon a given plane, the code must retrieve the index of that plane that has been clicked upon, before retrieving the index of the face(s) that were affected. This then has other connotations, as the mesh of a primitive box designed by Lingo code, has six planes, if the requirement is that the geometry of the box is deformed as the user clicks on a given plane, what happens when the user clicks on or around an area where two planes meet? This point is best illustrated by figure 5.11 below.

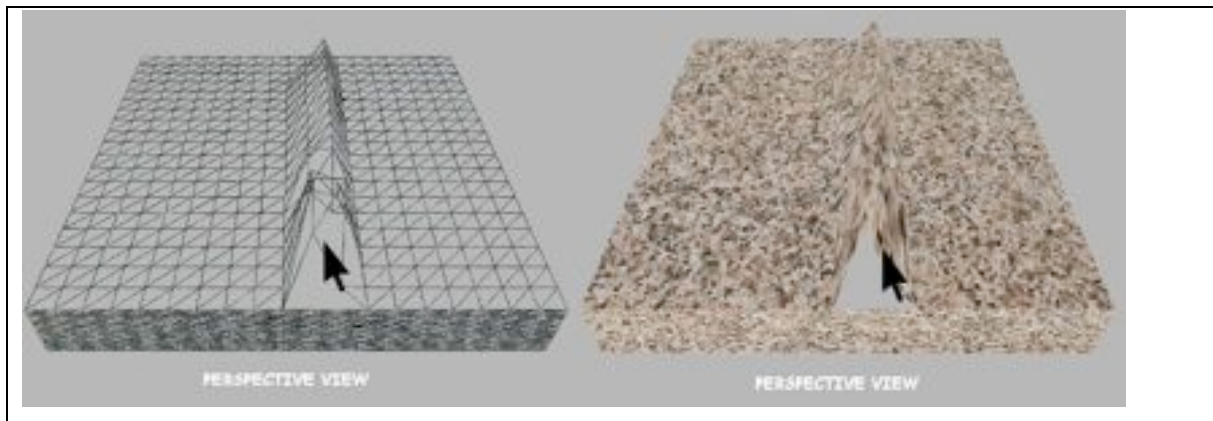


Figure 5.11 Plane separation

The diagram above illustrates the issues of plane separation, as now the affected plane has had its geometry altered so much that it no longer connects with the other planes which comprise the box model. This creates the impression that the object is now no longer solid. This issue seemed impossible to solve, and was not resolved until much later in the project's development. The next sections explain how the collision detection aspect of the simulator was created.

5.2 Collision Detection

Collision detection is one of the largest subjects in 3D programming. Director provides certain built-in tools for dealing with collision detection, however, these functions needed to be altered to best suit this project using vector maths.

“Collision detection is the process of determining whether two polygons in a 3D environment will intersect, and taking the appropriate action depending on the answer.” [17]

There are three distinct ways of managing collision detection within the Director environment, the first is to use the built in collision modifier, the second is to implement a form of boundary management, and the third is to use the Havok Xtra plug-in which has been discussed in section 5.4. Director's Collision Modifier provides a basic level of collision detection, but is extremely slow and has performance implications on the project. Firstly both plates must be assigned their own collision detector, when a collision occurs a handler is

called to perform some kind of response. The initial experiments for this show how the problem of plane separation starts to inhibit the performance of the simulation. This is demonstrated in figure 5.12 below.

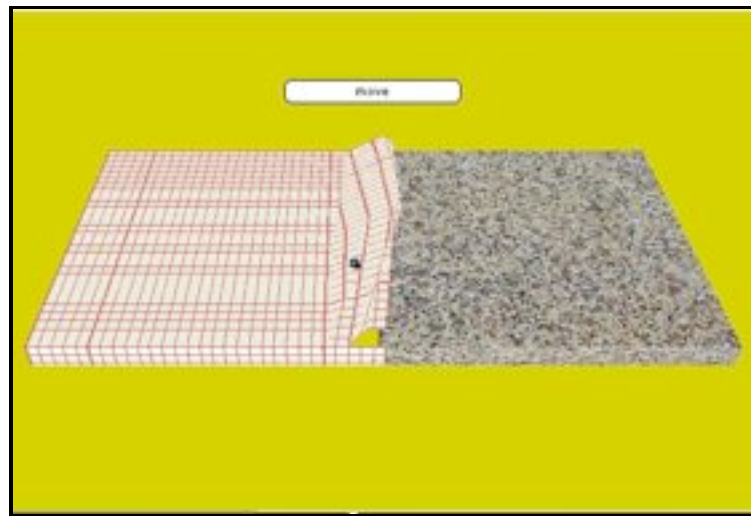


Figure 5.12 Plane separation with Collision Detection

It can be seen that as the collision is detected, the response takes action upon the top plane of the box model. This causes a separation of the model's planes. To overcome this issue, the initial experiments focused around locating those areas in which two planes will meet, i.e. at the corners. This proved extremely costly in computational time. Later experiments worked around an attempt to bind the mesh together so that the primitive box becomes one solid complete model. This led to the solution, thus far the collision detection and mesh deform functions have been invoked upon primitive models created within director using lingo. A shockwave 3D file that has been created in 3DS Max becomes one complete mesh upon exportation. This instantly solves the problem of plane separation. Using Exported models from 3DS Max, was the basis of the next set of experiments. An example of the type of result is shown below in figure 5.13.

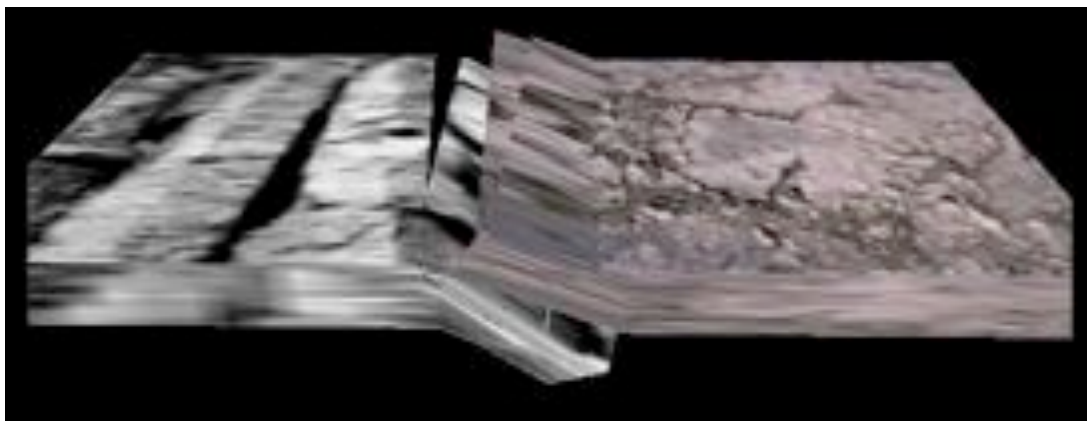


Figure 5.13 Plane separation Resolved

The resolution of the plane separation issue was a crucial step in a positive direction. From here there are two important issues to tackle in order to complete the simulation section of the project; these were to add

interactivity, and to improve the quality of the images which represent the tectonic plates. To improve the images was a task which had to be carried out in 3D Max, much time is spent working on different models, and trying to achieve a balance between high quality images and still maintain a high level of performance from the system. Some serious performance issues soon became apparent when dealing with higher quality images with a much higher polygon count. The simulation would now run up to 10 times slower, when used with the higher detailed images. This made the system impossible to use unless some other aspect of the system could be compromised. The most computationally expensive aspect of this simulation is the calculations involved in the collision detection, and although an important part of the simulation, it does not have to be exact in its results. It soon became clear that it was this area of the system that must be compromised in order to optimise the speed and usability of the simulator. The boundary management system sends out a message when a collision has just occurred within the environment. The mode with which this system works is of paramount importance to the system's performance. With a complex model such as the ones exported from 3D Max, if the collision mode uses mesh detection, then the collision modifier will check whether the actual model geometry has collided with any other geometry per frame. If the collision mode is altered to say that of a bounding sphere, surrounding the complex geometry, the collision modifier will work much faster but will be less precise. After trying this method it was decided that this was the best way to advance, as the speed and optimisation that is gained from doing this, far outweighs the cost of losing accurate collision detection.

Another conceptual hurdle which must be overcome with Director's collision detection modifier is the retrieval of the collision data. It is well founded that once a model's collision data have been retrieved, that model can no longer move within the scene.

“In order for the PointOfContact and collisionNormal information to be available, the resolve property for both models must be set to true. If the resolve is set to true, the models will no be able to move after the collision has been resolved.” [18]

The technique that was used to overcome this was to quickly disable and then re-enable the collision modifier after the collision has occurred, and the collision information has been retrieved. This enables the models in the simulation to move again, whilst still gaining access to the collision modifier. This involved precise timing, in order for the system to receive the correct collision data.

5.3 Creating Interactivity

As explained in section 3.1 interactivity is a key principle upon which this project is built. The simulator itself would be useless unless it has strong interactive characteristics. It was thought that the best way to do this would be to create an algorithm upon which the results of the collision are calculated, and for the user to be able to manipulate variables which alter this algorithm, thus a new resultant formation will be produced upon each alteration of the algorithm. There are three major inputs which are included in the movement of a tectonic plate.

“The plates-driving force is the slow convectonal movement of hot, softened mantle that lies below the rigid plates.” [19]

Although the simulator does not animate this convectonal current, it does seek to interpret the resultant force that it creates; the user is able to adjust the magnitude of this current in effect, altering the speed and force of the plate’s movement. The Density of each tectonic plate is also a very important factor in the creation of mountains. It is the oceanic crust which is denser and thus subducts underneath the lighter weight continental crust. This fact is emulated in the simulator, as the user is not able to make the oceanic plate less dense than that of the continental. If they do attempt this, then they are prompted that the simulation cannot continue, as illustrated in figure 5.13.1 below.

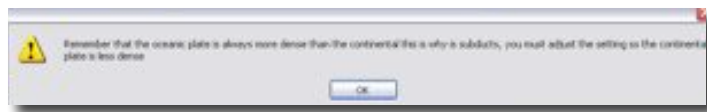


Figure 5.13.1 Plate density error

This is part of the learning process, as it is not the aim of the simulator to allow users to operate it without thinking about what they are trying to achieve. The final input that the user is able to manipulate is the rate of movement, the average rate of movement for a continental crust is 2cm per year, and that of an oceanic crust is as much as 5cm per year. This adjustment will alter the distance the plates will move between each step. After the user has entered the various inputs, the program manipulates the values before it executes the code. The rate of movement is not altered by the program, but the convection force and the rock density, are linked in nature, and so must be intrinsically linked within the program, the section below explains the algorithm that is used.

5.4 Simulation inputs

The maximum density that can be entered is 3.5gm per cubic meter, this value was chosen because the average density of an oceanic plate is 3 gm per cubic meter. It was clear when testing the simulator that the convectional current behind the force of the plate's movements must be an exponentially proportional relationship to the Density of the plate. The graph in figure 5.14 illustrates this relationship with the software.

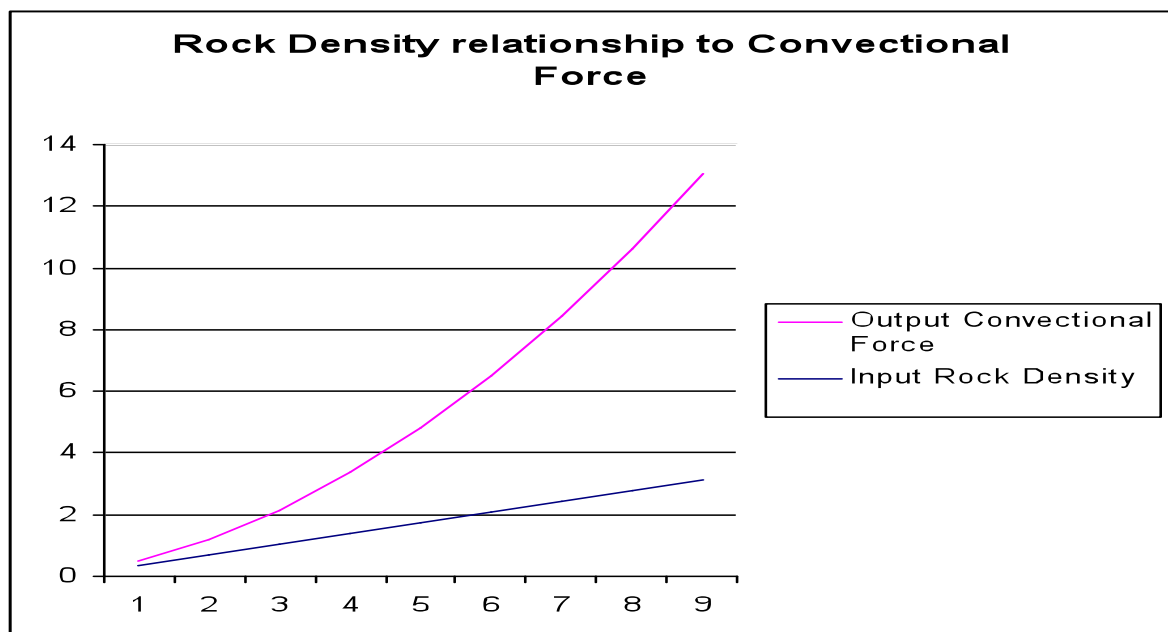


Figure 5.14 Rock Densities and Convectional Force

This relationship is created by the following block of code, where POceanicDensity is the input value for the oceanic plate, and in Density is the new input from the slider:

```
pOceanicDensity = 4.5 - inDensity
--inverts input density, as explained in next section

pOceanicforce = inforce * (4.5-inDensity)
--multiplies the input force, by the result of the maximum density(4.5)
minus the input density. This is in effect the same as multiplying it
by pOceanicDensity
```

However it is not as simple as this graph might suggest. In order for the user to be able to enter a value for each variable they will interact with a slider bar as shown below in figure 5.15.



Figure 5.15 Interactive sliders

As one might expect, sliding the bar from left to right should increase the value that it represents, this posed a problem as increasing the rock density should in effect create a more restricted movement of the plate, put in simple terms the greater the plate's density the less it should be deformed. If the value entered into the density slider above is put directly into the simulation, then the opposite would seem true, and plate with a large density, will seem to deform more than one with a low density, as the values the system is using to calculate the simulation are greater. In order to overcome this, a simple conversion is performed upon the input value of the rock density, this number is then converted back after each frame of the simulation has been calculated, so the user can see the feedback without confusion. This is shown in the following block of code:

```
--Conversion of density value for calculation purposes
pOceanicDensity = 4.5 - inDensity

--conversion back to original input for feedback purposes, resultant
value then put into text box ("oceanicDensity")
put ((4.5 + inDensity) - pOceanicDensity)/2 into
member("oceanicDensity")
```

The graph in figure 5.16 shows the relationship between the density that is entered and the density that is used to perform the calculations.

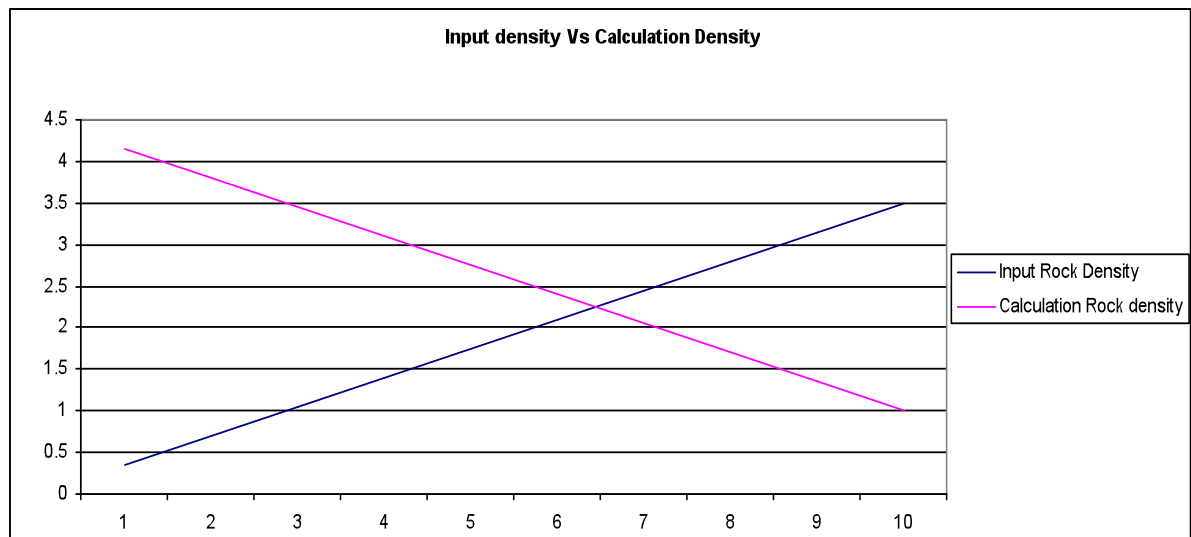


Figure 5.16 Input Rock Density Vs Calculation Density

5.5 Radio button system and plate textures

It is possible within this simulation to change the texture of the two tectonic plates, and the various input values which affect their performance within the simulation. This is done via a radio button system, whereby the user selects which plate to edit by selecting their associated radio button. Once the user changes their selection and reverts back to the unselected radio button, the variables they have already set, are “remembered”, and the values of the sliders are updated to represent the values that were previously set. The code for these radio buttons are in the appendices 37. The different plate textures are created each time the simulation is reset. Each texture represents a different kind of rock such as basalt or granite; there is also a grid texture which will enable the users to see clearly the geometry of the plates. Figure 5.17 shows the four textures that are available. The code below shows how the textures were created and assigned to each model.

```
--create a new shader
shd1 = scene.newshader("gridshader1", #standard)
shd2 = scene.newshader("gridshader2", #standard)
shd3 = scene.newshader("gridshader3", #standard)
--create 1st new texture
txt1 = scene.newtexture("gridtexture1", #fromcastmember,
member("basalt"))
txt1.nearfiltering = false
txt1.quality = #low
txt1.renderformat = #rgba4444-- to optimise performance we allow
render format not to store alpha information
--apply the texture to shader1
shd1.texture = txt1
--create 2nd new texture
txt2 = scene.newtexture("gridtexture2", #fromcastmember,
member("granite"))
txt2.nearfiltering = false
txt2.quality = #low
txt2.renderformat = #rgba4444
--apply the texture to shader2
shd2.texture = txt2
--create 3rd new texture
txt3 = scene.newtexture("gridtexture3", #fromcastmember,
member("grid"))
txt3.nearfiltering = false
txt3.quality = #low
txt3.renderformat = #rgba4444
shd3.renderstyle = #wire
--apply the texture to shader3
shd3.texture = txt3
```

Firstly a shader is created, given a name and type and assigned to the scene, this shader is then added to a texture, which has been specified as low quality and given a specific render format for optimisation purposes. The texture is then ready to be assigned to a model within the scene.

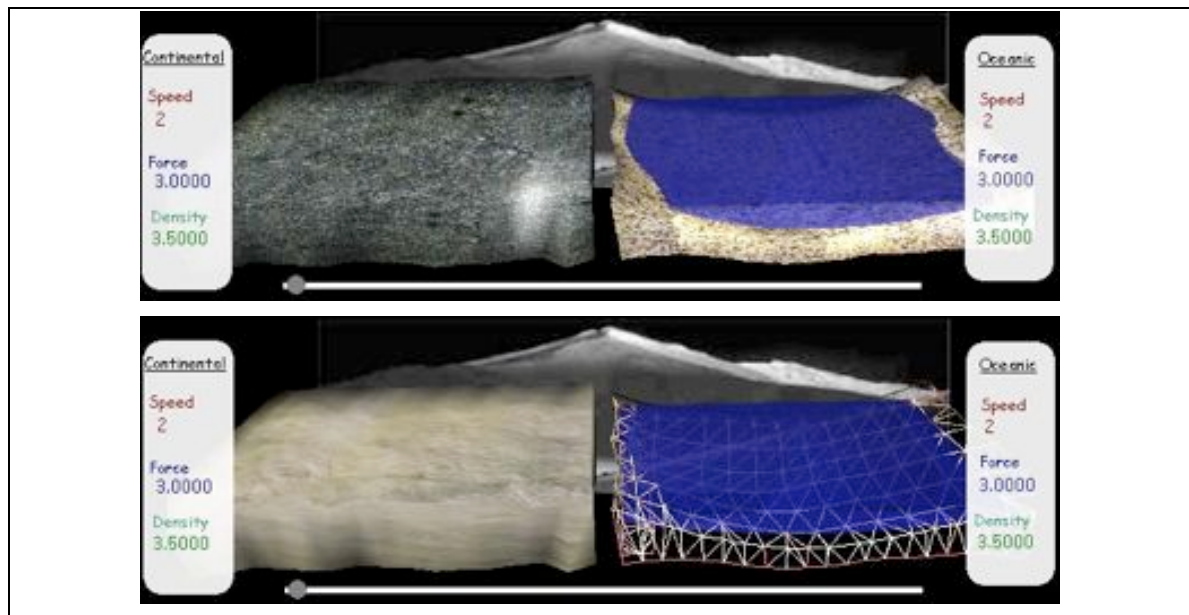


Figure 5.17 Various textures

5.6 Building a Keyframed animation with lingo

Extensive research showed that there is no pre-built Director function allowing the author to create animations at run-time using lingo.

“Animations cannot be created at runtime using Lingo, they must be authored in a 3D modelling package like 3D Max or Maya and be imported.” [19]

Like many programming obstructions, there is at least one way around this problem. The method that was used to overcome this was somewhat ad-hock, but the result works well. An animation had to be built from the simulation, which can be rewound and played for the student to be able to study. This is done quite simply by taking a snapshot of the stage after each step in the simulation, each snapshot is saved to a movie cast, and later displayed in the order at which they were taken, this creates the impression of a movie. The animation can then be played back at varying speeds, and will play much smoother than the original simulation, as there is very little code that needs to be executed in order to play the movie. The code below demonstrates how the snap shots are taken, and how the cast members are deleted when a new animation needs to be saved.

```
-- creates a new bitmap image of the stage and places it into a cast
member place holder
on makeBitmap
    NewBitmap = new(#bitmap, castLib "StagePictures")
```

```

--creates new empty bitmap as a placeholder
NewBitmap.picture = the stage.picture
--puts a screen print of the stage into the bitmap place holder
if the timer > 180 then
    castLib( "StagePictures").save()
--the system will stop takes a screen shot after 180 miliseconds
end if

end

-- deletes any previously saved bitmaps, ready for new sequence
on deletecast
    finish = 21
    repeat with i = 2 to finish
        erase member i of castLib "StagePictures"
    end repeat
end on deletecast

```

The #Make Bitmap() function is invoked at each step of the simulation, as the code below demonstrates, this way a series of images are created.

```

on exitFrame me
    repeat with x=1 to 20
        go "sim" --stay on this frame
        scene.model[2].translate(2,0,0)--move continental plate
        scene.model[3].translate(-2,0,0)--move oceanic plate
        scene.model[4].translate(-2,0,0)--move water
        sprite(56).loc = sprite(56).loc +point(20,0)--this is where snap
shot is taken
        makeBitmap -- Calls the make bitmap function shown above

    end repeat
    go to "endSim"
end

```

The #deletecast() function is called as soon as the ‘Simulate’ button is pressed.

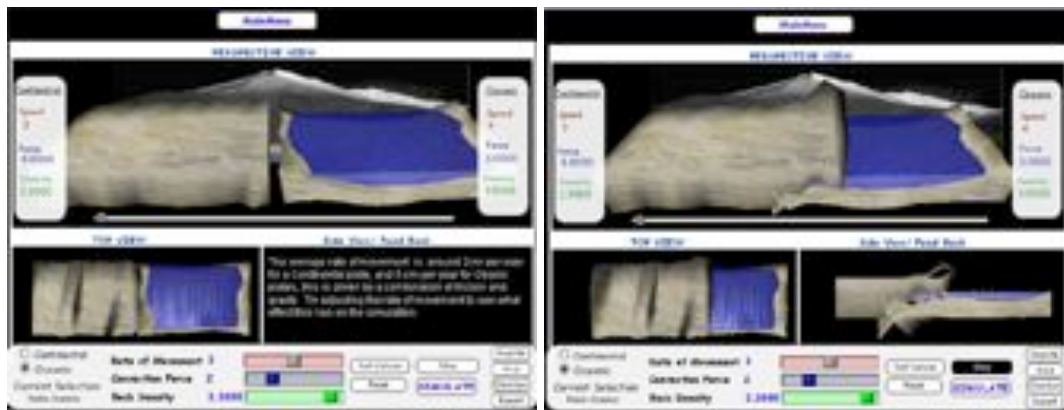
```

on mouseUp me
    deletecast
    go "sim"
end
set the member of sprite 2 to member 2 of castLib "stagePictures"

```

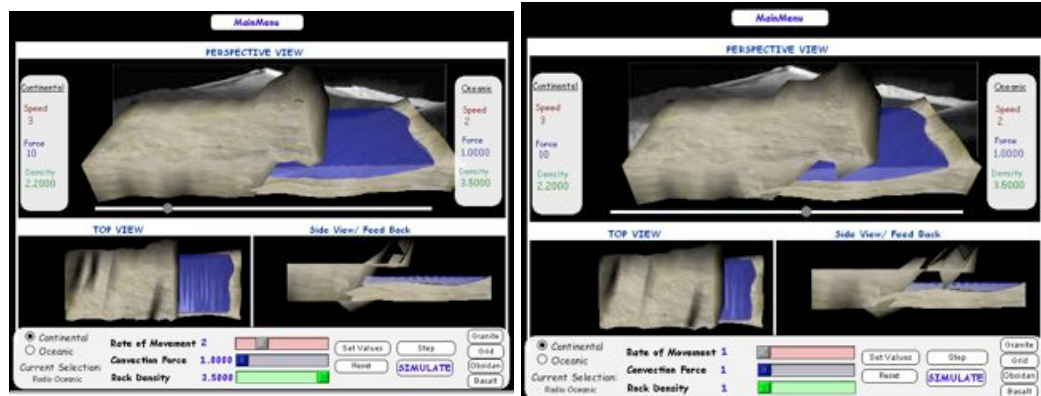
-- This line of code simply loads the first of the snap shots onto the stage.

Figure 5.18 shows some typical interactions with the completed simulation.



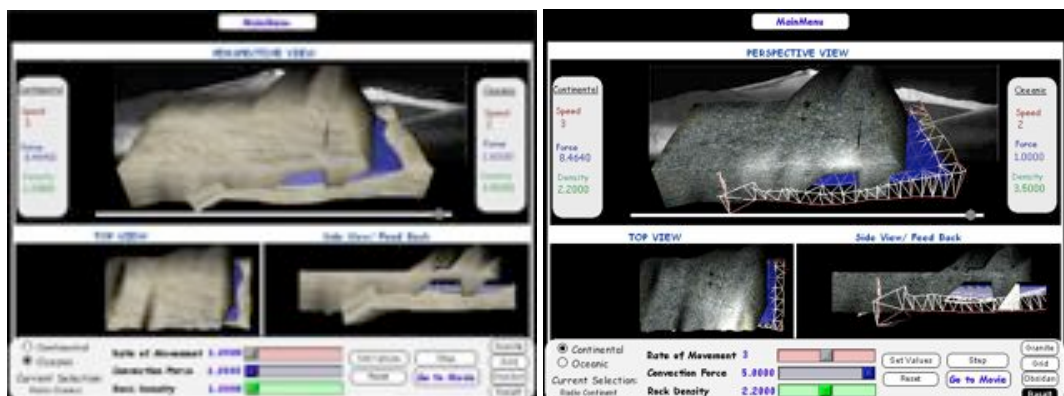
User alters setup values, and click on “set Values”

User can use the step button to step slowly through the simulation and alter the variables as they wish



User Clicks on “Simulate”, and player head begins to move as plates begin to deform.

Half way through Simulation



Simulation has finished, user can click on
“Go to movie to playback the animation.

User changes textures.

5.7 Interactivity outside the simulator

There are many other aspects of this project which contribute to the interactive learning experience of the tutorial. The separation of Pangaea, is an important theory in plate tectonics and one which must be understood in order to grasp the fundamental concepts behind how the earth is formed, it is also a key learning topic within the AQU GCSE Specification As explained in Figure 2.1 (Tectonic activity within education). In order to incorporate this concept whilst keeping the interactivity theme, it was thought that an interactive timeline would allow the user to understand how the separation occurred through the passing of billions of years. Figure 5.19 below demonstrates how, scrolling the bar from left to right gives the impression of a user controlled animation.

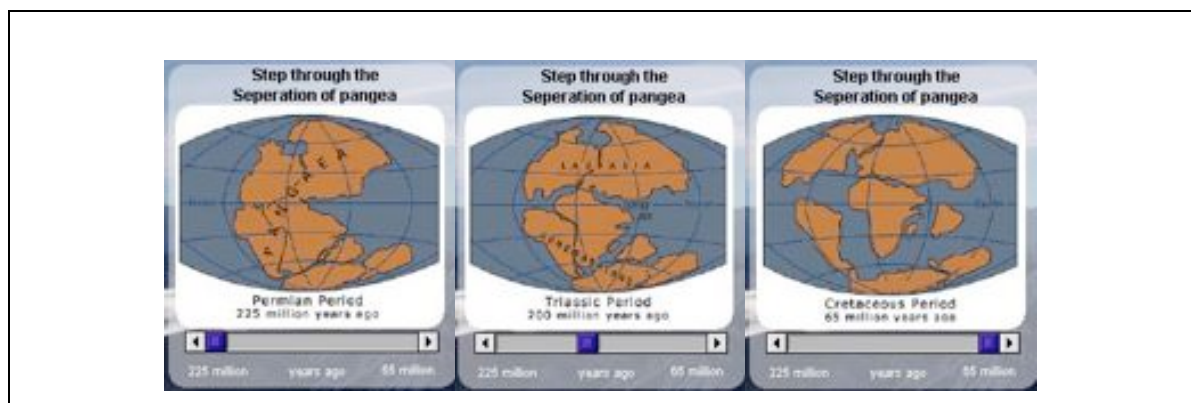


Figure 5.19 Pangaea Timeline

The method that was used to retrieve the current position of the slider is shown below, it checks that the input value (pValue) is equal to current value, if not then the code sets the thumb slide to the corresponding position.

```
on getCurrent4 me, val
    if pValue = val then exit
    pValue = min(max(val, pMinValue), pMaxValue)
    pThumbOffset = GetButtonOffset(pValue)
    me.DrawElement()
    sendSprite(me.spriteNum, #ChangeValue, pValue, 1)
    updateStage
end
```

The current position information is then sent to a handler which decides which image must be visible and which must not. The script for this is shown in the appendices 15.

The project uses the concepts of 3D graphics throughout the implementation process, the section on “more about fold mountains” takes the user to a 3D Diagram of a fold mountain, the user is able to rotate the model, zoom in and out and to interact by clicking on various parts of the model to produce a detailed account of its role in the process of plate tectonics. A new window appears when the user clicks on the “more about Fold

Mountains” section which is in fact a separate director movie, designed to look like a television screen, with various controls on the left hand side. The code used to open the movie in a new window is shown here:

```
window("diagram.dir").open()  
window("diagram.dir").title = "3dDiagram"
```

The Figures that follow, illustrate how the user is able to interact with the model. Image ‘A’ shows the model in its reset state, image ‘B’ illustrates the rotate capabilities. Image ‘C’ demonstrates the Zoom tool. Images A-C make use of some of Directors pre-built behaviours, a modification to the code was needed to interpret which tool to invoke from these three inbuilt behaviours. In order to do this, each time the user clicks on one of the tools, a secret action field is set to a string which represents that tool. The modification of the pre-build director behaviours simply checks what the action field is currently set to, and invokes the corresponding method. The code below shows how the action field is set, and also how the cursor images were changed, without using director’s pre-built cursor change behaviour.

```
global action  
on mouseUp me  
  action = "rotate" --different action for each tool  
  cursor 2 --each number represents a cursor type, different for each  
  tool  
end
```

Image ‘D’ Shows how the user is able to click on different layers of the model to find out information about them, notice also that when the layer is clicked on, its texture becomes that of a transparent wireframe (appendices 20). This enables the user to distinguish the various layers of the model. A difficult aspect of the code behind this interaction was changing the cursor’s image as the user clicks on a new tool, even more problematic, was keeping the new cursor image on display whilst the mouse is within the model areas space. The code used to solve this problem is shown below:

```
global action  
property pSprite  
property pCursor  
on mousewithin me  
  pSprite = sprite(me.spriteNum)  
  if action = "rotate" then -- check action field  
    pCurspr=2 --changes cursor  
    pSprite.cursor = pCursor  
  else if action = "zoomin" then  
    pCurspr=302  
  else if action = "zoomout" then  
    pCurspr=303  
  end if
```

end

This code simply checks the action field, and sets the pCursor value to the number that corresponds to the appropriate symbol, e.g. value 2 will be represented as a crosshairs symbol. Figure 5.20 shows different interactions with the 3D diagram.

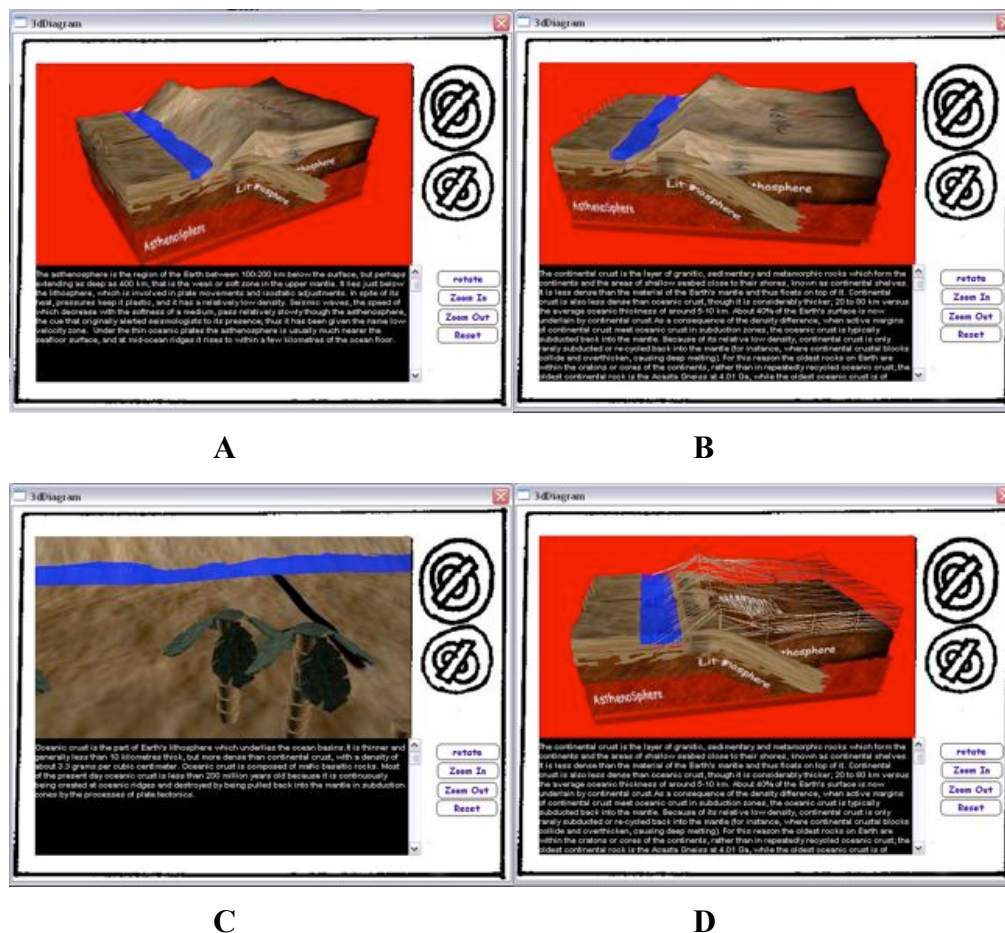


Figure 5.20 Interactive 3D Model

5.8 Picture slide show

As the creation of this tutorial was inspired largely by a personal interest in its subject, it seemed fitting to create a section which clearly depicts different mountains, and categorises them by the processes by which they were created. This should enable the user to visualise in real terms, the scale and magnificence of what they are learning about. It was done in the form of a slide show and is displayed within a frame of a television set, the user is able to skip forwards and backwards through the photos, and pressing the off button will close the window completely.



Figure 5.21 Picture slide shows

The code used to open this movie within a new window shown below:

```
on mouseUP me
    window("mountainImages.dir").open()
    window("mountainImages.dir").title = "Pictures"
end
```

The movie uses Directors inbuilt built behaviours to create a typewriter effect, for the text, and the fade in and out behaviours give smooth transitions between each picture.

5.9 Animations

To add a further dimension to the interactive learning experience, the use of 3D animations was a crucial part of the tutorial, the user is able to play back movies which illustrate the three most common types of mountain formation. These can be rewound and re-played, whilst the user reads about exactly how these occurrences happen around the world. These animations were created in 3D studio max, and exported as AVI files. The code which gives the user playback control is shown below:

```
on mouseUp me
    sprite(4).movietime = 0 -- Rewind to start
end

on mouseUp me
    sprite(4).movieRate = 0 -- Pause
end

on mouseUp me
    sprite(4).movieRate = 0.5 -- Play (at half speed)
```

end



Figure 5.22 Animations

5.10 Navigation

In order to ensure the tutorial is consistent throughout the software, the same text size and font has been used throughout. Another important feature is to ensure that the text always appears within the same frame, this is done using code which automatically loads the text into a pre-set place holder. The text is stored outside the program as an RTF (rich text format) file, and loaded into the appropriate text box or title box, as soon as the play head enters a given frame, or once a button has been depressed. The code below shows how the text is loaded into member (193) which is the empty text box place holder. The code also ensures that the text box always has the same dimensions, is scrollable and that the font is size 12 and loads the title “Tectonic plates” into a title placeholder.

```
on mouseUp me
  go to "learn"
  member(193).filename = "tectonics.rtf"
  member(193).width = 380
  member(193).height = 320
  member(193).boxType = #scroll
  member(193).fontsize = 12
  member(133).text = "Tectonic Plates"
end
```

This method of editing the text outside the program and dynamically loading it into the same frame, guarantees that the text is always in the same location and will always have the same properties. The result of this is that the program is efficient consistent and easy to use, as the user will quickly get used to the standardised layout.

6 Testing

A systematic check routine must be done in order to test that each element of the system works. This is carried out using an ad-hoc checklist for each page of the tutorial, each command and navigation button is tested for the correct action, the simulation is checked both for correct typical use and incorrect usage. The results are shown in appendices 4. They demonstrate that the system works well as almost all tests were successful. There were only two areas where the system was found to be unsatisfactory. The first unsuccessful check was that of the intro movie as it jumps slightly during playback. The solution to this lies in the VRAM optimisation extensions explained in the conclusion of this report, and is part of a greater optimisation issue that could be managed in future extensions. The second Faulty test occurs when the user clicks the reset button in the simulator, both the feedback values and the slider bars should return to zero but values on feedback panel remain the same. This creates issues with feedback, and could confuse the user. This problem could not be resolved within the time scale, as it requires complex analysis of the passing of data between the sliders and the feedback panel. Future implementations would have to assess this data flow. After establishing that the system as a whole, does indeed function correctly, a series of user evaluations must next be carried out, to understand its usability.

7. Evaluation

In order to establish the extent to which the Mountain Formation Tutorial is usable and successful in fulfilling its requirements, a combination of user testing and evaluations must be carried out. The first form of testing that has been done is a controlled investigation of a specific aspect of the interface.

7.1 Task Specific User Testing

A specific task that must be investigated is the simulation section of the tutorial to establish to what extent user's interaction yields desired results. The user first reads the instructions shown in appendices 6 to familiarize themselves with how to use the simulator. They are then asked to perform 3 simple tasks, the first is to change the textures of the continental plates, the second is to setup and create a step through simulation and the third, to create a complete automatic simulation. Each user is timed in completing all three tasks, and the number of user errors that each user encountered was recorded. A user error was defined for the purposes of this test, as any result given by the system that the user did not understand or predict. It was the user's task to record any such errors, and explain why they thought them as such. The aim of this test is to assess the performance of a user's interaction with the simulation to gain a picture of how well the system adapts to the needs and preconceptions of different users, and to illustrate how well the system teaches and explains to the user how to do the task in hand. The results of the test are shown below in figure 6.1.

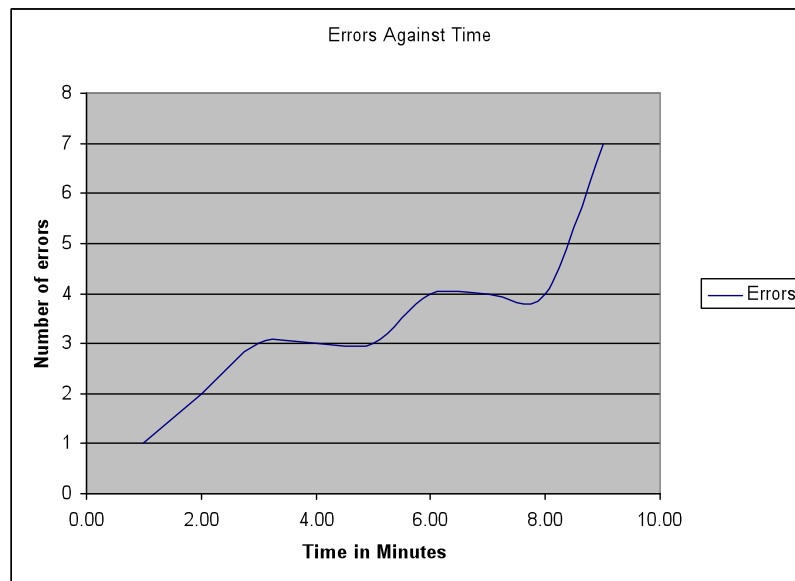


Figure7.1 User Errors

The graph shows that the higher the amount of errors a user encounters, the longer they take to complete the tasks. This would suggest that for some users, the program is not sympathetic to their learning types or their expectations of the system. This test suggests that the system does not have enough feedback or precautionary messages, therefore if a user encounters an error early on in the interaction, they are unsure how to recover from that error, and are invariably lead into more problems. This hypothesis is re-enforced from the further testing shown in the next sections.

7.2 Testing Requirement Analysis

In order to determine how well the Mountain Formation Tutorial meets its user requirements set out in the initial planning stages, a series of systematic checks must be made to guarantee that the software does what is required; the checks shown in appendices 7, layout the primary and secondary objectives, and breaks them down into subtasks, these subtasks are then assigned a tick if they have been met, or a cross if they have not. From establishing how many of the subtasks have been met it is possible to justify to what extent the requirements have been fulfilled, and even to determine a percentage of fulfilment, although this is purely conceptual. One Requirement that is not fully fulfilled is:

“To Prompt the user to enter variable values into the program such as: Force(N); Mass; time; rock type. “

The user is not prompted to alter the values for the three variables into the system before starting the simulation. This is because it was thought to be a little too restrictive to constantly remind the user to enter in new values each time the simulation is re-set, some users are already aware of this, and feel comfortable in controlling the simulation at their own pace. However, other users who are less confident with using programs of this nature, encountered user errors at this stage of the simulation and they did not feel they had been given adequate guidance. Perhaps a solution to this would be to have a help document or ‘wizard’ within the simulation, with which the can be guided through the set up process, and disable once they feel ‘safe’.

A second requirement which was found not to be fully fulfilled was:

“To Make the program interactive and enjoyable to use.”

The only way this was found to have failed was the exclusion of any sound media within the program. The tutorial could have benefited from a use of sound/music, in various sections, as some users will respond better to this form of media than to just reading for example. However, all things taken into account, the Tutorial is diverse in its usage of interactive multimedia, and most users felt that the lack of sound did not inhibit their learning experience.

The third objective which was not fully fulfilled within the scope of the program was:

“The user must be able to control the camera view of the animation as it is in motion.”

This objective was in fact fundamentally flawed and was not possible to implement within the time scale that was given. Director has pre-built behaviours which enable the user to control camera positions. These behaviours can be used in conjunction with any 3D model which does not change state whilst the program is running. If a model does change state, and needs to be re-set to its original state at any point during run-time (as is the case within the Mountain Formation Tutorial), then the camera behaviours that are bound to that model, will encounter an error as the process of re-setting the model involves deleting it and re-importing. This is true with any of the Director pre-built behaviours, there for all actions relating to the simulation had to be "hard coded" without using any of the prebuilt behaviours.

7.3 Open ended User study

In order to attain some more open ended feedback, the user test group were instructed to complete a series of questionnaires as they went through the program, this gives each user a chance to express their own views on the software, and gives an insight into what the users are thinking as they use it. Ideally these questionnaires, would be used to set up an iterative evaluation in which the views expressed are implemented within the program and then resubmitted to the user test group, until such time that the users are happy with the program. The feedback attained from these questionnaires can be used to understand what further implementations could be considered in future extensions of the program. Actual questionnaires can be found in the appendices number 5.

What has become clear from the open ended user study is firstly that each user felt that they learned from using the Mountain Formation Tutorial, and that it was on the whole a pleasurable learning experience. It is also clear that the simulator was a success with all the users feeling that it was the most useful learning tool within the tutorial. Generally the design of the software was well received, and thought to be pleasing. The problems that are apparent are once again concerning the system guidance and feedback, with some users feeling that the system does not give adequate safety from errors, and does not provide enough user documentation on how to interact with the tutorial.

7.4 Targeted User study

The Targeted user study establishes a more directed feedback, and asks the users to think about certain heuristic aspects of the system, establishing a scaled evaluation of these aspects. This gives a direct insight as to how the users gauge the efficiency, usability, and aesthetics of the Mountain Formation Tutorial. The results of the questionnaire can be seen in appendices 70. They demonstrate that the software scores highly in its informability, and aesthetics, with all users giving good feedback for what they have learned from the tutorial, the tests also point out that there is not enough safety checking, and error recovery within the program. A more detailed analysis can be drawn, by comparing the user feedback against Nielsen principles of heuristics: [25]

Visibility of system status

The system does not always keep users informed about what is going on with appropriate feedback, and some of the feedback that is given is difficult to understand. It could benefit from loading screens, and wizards, to help the user to set up the simulation.

Match between system and the real world

The system has been found to have a logical order, and is consistent in its design.

User control and freedom

The lack of an undo function means that the user must go through an extended dialogue if they make a mistake, this limits the user's freedom somewhat.

Consistency and standards

Users do not have to wonder whether different words, situations, or actions mean the same thing. As they find the program to be consistent and follow internal platform conventions.

Error prevention

This seems to have been the most common problem for all users, as the system does not take enough precautions to prevent errors. This could be resolved by taking more time to eliminate error prone conditions or by checking for errors more often within the simulator and presenting the user with confirmation options.

Recognition rather than recall

The feedback suggests that the user's memory load has been minimised. With the use of objects, actions, and options clearly visible they do not have to remember information from one part of the dialogue to another. Instructions for use of the system are visible and easily retrievable whenever appropriate.

Aesthetic and minimalist design

The design and aesthetics of the program were highly rated with dialogues not containing information which is irrelevant, and the layout clear and concise.

Help users recognize, diagnose, and recover from errors

Error messages were found to be expressed in a way which did not explain precisely what the problems were or how to recover from them with a constructively suggested solution.

8. Conclusion

After extensive evaluations and testing it is found that the Mountain Formation Tutorial is a largely effective, aesthetic and informative program, fulfilling its objectives and requirements to a satisfactory level.

The research aspects to this project have been largely successful with an exploration into a wide range of technologies that can be used to create and manipulate 3D simulations across the two software platforms.

Through a series of experiments and tests, it has been discovered that the best way to interlink and manipulate the models between the two programs, is to create the geometry of the model in 3D Max, and export this geometry as a shockwave file. The #MeshDeform modifier within Director can then be used to manipulate the model in almost anyway that is required.

A crucial aim of this body of work is to investigate the possibilities of creating an interactive simulation by interlinking two programs (Director and 3D Max). This has been an exploration into a somewhat uncharted realm of multimedia design, it is well known that Director MX is a powerful tool to view, build and control the movement of 3D graphics, but a different truth is revealed when examining specific aspects of its 3D capabilities. When working with pre-built and pre-rendered 3D models, Director's options for 3D manipulation become rather limited if the 3D models are altered and reset within run-time. When this is the case, the pre-built tools within Director for model manipulation cannot be used. As a result any actions that must be invoked in order to alter a models geometry must be laboriously coded by the programmer, therefore Director loses its easy interface advantages. That said, it is still possible to interface between the two programs and as shown by this project, it can be done with a vast degree of customization, and yield desirable results. In seeking to investigate and push the boundaries of Directors capabilities to deal with pre-rendered models from 3D Max, much attention has been focussed on the successful research, experiments and implantation of the simulator. This has resulted in an oversight of the extent to which a novice user might need to refer to help and instructional documentation. In order to address and improve this issue of a lack of visibility of the system's status and better error recovery messages, an iterative evaluation would need to take place, to ensure these cracks in the program's usability are corrected.

The secondary objectives that were proposed for the system were somewhat ill-informed, and simply not possible to implement within the time scale. It was realised early on in the implementation that these extensions could not be carried out, therefore much thought was put into the proposal and implementation of other possible extensions such as the interactive 3D diagram, and the 3D mountain sculptor. These extensions give the program an added interactivity. Future extension for the program would be focussed mainly around the usability of the simulator, and the reductions and prevention of user errors. Other extensions would be beneficial such as a topographical map which the user could switch on and off, in order to see how the movements of the plate affect the height of the mountain. Another part of the simulation which could be extended, was the reactivity of the water. In reality, as a continental plate pushes against that of an oceanic, the water levels of ocean will rise as it is pushed back. Currently the simulator does not simulate this, different methods were tried, particularly concerning the #DeleteVertex function within director, although none were adequately successful. In future extensions of the program, it would be

desirable to fix this issue, and animate the movement of the oceans, in response to continental plate movements.

Alternate methodologies could have been used to implement the software, although it was found that a combination of Shockwave exportation and the #MeshDeform function serves its purpose well, further experimentation with the bone style animations could also yield a desirable result. This idea of taking a methodology which is specifically designed for the emulation of a bipedal skeleton, and adapting it for the purposes of this project is an inviting one as it investigates the boundaries of the bones integration system of Director, in the same way that this project has sought to extend the limitations of the #MeshDeform function.

The software has undergone some crucial optimization extensions but further improvements can still be made. Two major areas for performance optimization are height and width of a 3D image and the size of the users desktop. This is because VRAM is allocated for both the desktop and the 3D images. Very large desktop sizes will require a considerable amount of VRAM. The 3D images will need to allocate an amount of VRAM that's dependant on the size of the image, the colour depth of the screen and the z-buffer for the active 3D renderer. The amount of VRAM needed for the image can often exceed the amount needed for the desktop. One key area of performance is preserving VRAM, and so a detailed analysis of controlling the image and the desktop could vastly optimise the program. It would also be useful to determine the amount of VRAM on the user's video card and/or to estimate the amount of VRAM the Director project requires. There are three aspects which dictate a projects VRAM requirement these are:

The screen	(Screen width * Screen height * Colour Depth)	VRAM screen requirements
3D Images	(Image width * Image height * colour depth)	Image colour requirements
	(Image width * Image height * zBuffer)	Image depth requirements
	Image colour + Image depth)	Total Image VRAM requirements
Textures	Texture width * Texture * colour depth)	Texture VRAM requirements

Once the projects VRAM requirements have been established alongside the Users VRAM capabilities, a number of steps can be taken to improve optimization, such as:

- Scaling down textures, as they are loaded based on the ratio between the amount of VRAM required and the amount present
- Use multiple external casts libraries with pre-made textures for different VRAM Configurations
- Force users into software rendering mode, with a scaled down version of the project
- Warn users tat they do not have enough VRAM

Implementing these optimisation extensions, alongside the extensions suggested for simulation would enhance the user's learning experience, and ensure the Program always runs efficiently.

9. References

- [1] *3D Studio Max® 7.0, discreet, Copyright © 1997-2004, Inc All rights reserve, Help documentation*
- [2] *Director, MX, © 1984-2002 Macromedia Inc All rights reserved*
- [3] *Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Computer_simulation, Accessed on 10/11/05*
- [4] *J.Westaway & E.Rawling, Geography Dept Qualifications and Curriculum Authority
“A new look for GCSE geography?” 2001*
- [5] *J.Westaway & E.Rawling, Geography Dept Qualifications and Curriculum Authority
“A new look for GCSE geography?” 2001*
- [6] *British Computer Society 2005, Code of Conduct, <http://www.bcs.org/server.php?show=nav.6030>
Accessed on 05/11/05*
- [7] *British Computer Society 2005, Code of Conduct <http://www.bcs.org/upload/pdf/cop.pdf>
Accessed on 05/11/05*
- [8] *AQA Examining board, General Certificate of Secondary Education Geography C 200,
<http://www.aqa.org.uk/qual/pdf/AQA-3033-3038-W-SP-07.pdf> accessed on 10/11/05*
- [9] *AQA Examining board, General Certificate of Secondary Education Geography C 200,
<http://www.aqa.org.uk/qual/pdf/AQA-3033-3038-W-SP-07.pdf> accessed on 12/11/05*
- [10] *David Leat, Geography--Study and teaching (Secondary), Thinking through geograph
Page 120*
- [11] *Richard Fothergil, Implications of new technology for the school curriculum, Kogan Page, 1988
Page 120*
- [12] *Adam Kane ForgeFX lead programmer, www.forgefx.com accessed on 12/11/05*
- [13] *University South Wales , Australia <http://www.fbe.unsw.edu.au/Learning/Director/3D/Havok/>
,last accessed: 6/3/2006*
- [14] *Macromedia official website
http://www.macromedia.com/support/director/work_3d/models_use_in_sw/models_use_in_sw07.htm, last accessed: 6/2/2006*
- [15] *3D Studio Max® 7.0, discreet, Copyright © 1997-2004, Inc All rights reserved, Help
Documentation*
- [16] *Copyright 1999-2005 Havok.com, <http://www.havok.com/>, accessed on 12/11/05*
- [17] *Paul Catanese, Directors Third Dimension 2002 Page 781*
- [18] *Washington University, 21-Feb-2001 18:52,
http://www.atmos.washington.edu/2001Q1/211/Group_projects/group_A_W01/what_makes_the_plates_move.ht, accessed on 15/03/06*
- [19] *Paul Catanese, Directors Third Dimension 2002 Page 785*

- [20] *University South Wales , Australia*
<http://www.fbe.unsw.edu.au/Learning/Director/3D/3DCastMembers.asp> ,accessed on 8/3/2006
- [21] *Lingo widgets Slider*
<http://www.ullala.at/experiments/ilwidgets/index.html>, accessed on 15/3/2006
- [22] *Lingo widgets Radio Button*
<http://www.ullala.at/experiments/ilwidgets/index.html>, accessed 13/3/2006
- [23] *Ben Bederson / Saul Greenberg, Graphical Screen Design*
<http://www.cs.umd.edu/class/fall2002/cmsc434-0201/notes8.pdf>, accessed 13/11/2005
- [24] *Moorland School, Lancashire*
<http://www.moorlandschool.co.uk/earth/tectonic.htm> , accessed 13/11/2005
- [25] *Jacobs Nielsen, Principles of Heuristics for User interaction*
http://www.useit.com/papers/heuristic/heuristic_list.html, accessed 10/3/2005

10. Bibliography

- *Director's Third Dimension - Paul Cananese, 2002, Que publishing*
- *David Leat, Thinking through geography, 2005, Nelson Thornes*
- *Implications of New Technology for the School Curriculum - Richard Fothergill, 1988, Kogan Page publishing*
- *Issues in Geography Teaching - Chris Fisher, Tony Binns - 2000 RoutledgeFalmer*

11. Figures

<i>Figure 2.1 Tectonic activity within education [10]</i>	10
<i>Figure 3.1 Types of Interactivity</i>	12
<i>Figure 3.2 Topographic Map Case Study Prentice Hall</i>	13
<i>Figure 3.3 Ocean Wave Case Study Prentice Hall</i>	13
<i>Figure 3.4 Case Study Moorland school</i>	15
<i>Figure 4.2 Final Design and navigation map</i>	17
<i>Figure 4.2 Start page front end design Mid-FI 1</i>	19
<i>Figure 4.3 Simulation page front end design Mid-FI 1</i>	19
<i>Figure 4.4 Simulation page front end design Mid-FI 2</i>	20
<i>Figure 4.5 Menu page design Mid-FI 2</i>	21
<i>Figure 4.6 Tectonic plates design Mid-FI 2</i>	21
<i>Figure 4.7 Animation design Mid-FI 2</i>	22
<i>Figure 5.1 Key Frame Interpolation</i>	23
<i>Figure 5.2 Bone animation</i>	25
<i>Figure 5.3 Rendered Bone animation</i>	25
<i>Figure 5.4 Space warps</i>	26
<i>Figure 5.5 Push and Wave Space warp</i>	26
<i>Figure 6.5 Shockwave file analysis</i>	27
<i>Figure 5.7 Detailed picking of a plane</i>	28
<i>Figure 5.8 Detailed picking Code</i>	29
<i>Figure 5.9 Construction of 3D model</i>	30
<i>Figure 5.10 3D polygon world</i>	30
<i>Figure 5.11 Plane separation</i>	31
<i>Figure 5.12 Plane separation with Collision Detection</i>	32
<i>Figure 5.13 Plane separation Resolved</i>	32
<i>Figure 5.13.1 Plate density error</i>	34
<i>Figure 5.14 Rock Densities and Convectional Force</i>	35
<i>Figure 5.15 Interactive sliders</i>	36
<i>Figure 5.16 Input Rock Density Vs Calculation Density</i>	36
<i>Figure 5.17 Various textures</i>	38
<i>Figure 5.18 Simulation Screen Shots</i>	41
<i>Figure 5.19 Pangaea Timeline</i>	42
<i>Figure 5.20 Interactive 3D Model</i>	44
<i>Figure 5.21 Picture slide shows</i>	45
<i>Figure 5.22 Animations</i>	46
<i>Figure 7.1 User Errors</i>	48

12. Appendices

1 Planning.....	58
2 Gantt chart.....	59
3 Email contacts with Forge FX.....	60
4 Ad-hoc Test.....	61
5 Open ended user study 6 Specific task user testing.....	65
6 Specific task user testing.....	66
7 Requirement Analysis Checklist	67
8 System Wide code and behaviours.....	68
9 Jump to lean and load Text.....	69
10 Jump to Simulator and load Text.....	69
11 Jump Mountain Sculptor.....	69
12 Jump to “fold” and load Text.....	69
13 Jump to Front Menu	69
14 Pangea Timeline Scroll Script.....	70
15 Pangea Display handler	70
16 Load Diagrams.dir	71
17 Jump to “animations”	71
18 Jump to “Lets see some pictures”	71
19 Jump to tutorial1.....	71
20 Image D, Pick layer, display text, change render style	72
21 Orbit camera.....	74
22 Orbit camera trigger.....	75
23 Check cursor symbol.....	75
24 Camera zoom in.....	75
25 Zoom camera trigger.....	76
26 Zoom camera out	77
27 Zoom out camera trigger.....	78
28 Reset Camera.....	78
29 Reset Camera Trigger.....	78
30 Button Navigations	79
31 Button Navigations	79
32 Button Navigations	79
33 Button Navigations	80
34 Button Navigations	80
35 Close slideshow	81
36 Button Navigations	81

37 Radio Button	82
38 Update rate of movement value	84
39 Update convection force value	84
40 Update rock density value	85
41 Set rate of Movement slider	86
42 Set rate of movement feedback	86
43 Set convection force slider.....	86
44 Set convection force feedback.....	86
45 Set rock density slider.....	86
46 Set density feedback.....	87
47 Set values Button.....	87
48 Reset Button	88
49 Step Button.....	88
50 Simulate Button.....	91
51 'Sim' Frame	91
52 Making a screen shot of simulation	92
53 Change texture to Basalt	92
54 change texture to Grid.....	92
55 change texture to obsidian.....	93
56 change texture to Basalt	94
57 Initialisation script.....	94
58 Play head script to call initialisation.....	96
59 Check 3DScene	96
60 Naming movie globals	97
61 Mouse over help text.....	97
62 Play animation.....	97
63 Deforming the Plate.....	97
64 Initializing scene.....	98
65 Create Box and Plane Primitive with lingo.....	99
66 Changing textures, granite basalt, and grid.....	100
67 Set up globals for movie	101
68 Check scene is ready.....	101
69 Reset button	101
70 Targeted User study.....	102

1 Planning

The Gantt chart shown in appendices 2 shows the detail of planning that has been put into the creation of this project. Each specific Task has been broken down into a series of executable steps, which have been defined in terms of their duration. The chart shows the expected duration of each task, and the progress thus far.

Background Research (plate tectonics)

The background research involves understanding how the movements of tectonic plates affect the creation of the earth's surface. The purpose of this research is to understand what kinds of different movements there are and what kinds of physics are involved.

Background Research (Director)

This is an important aspect of the creation process. As a programmer I have never used Director before, although I am aware of some of its capabilities. This research is intended to understand how Director works, and eventually to find what it is and is not capable of in relation to the project.

Investigate how to alter 3d animations in lingo

This task was ill informed, after investigation into this subject it is understood that if the animation is rendered before being imported into Director, Lingo can have little effect upon the animation other than simple control mechanisms, e.g. Stop, rewind, pause etc. However lingo can be used to create 3D graphics, but this would be far more time consuming than using a modelling package such as 3DS max, and to a less attractive effect. It has been discovered that the Havok physics engine may be more desirable to use for the purposes of creating and manipulating real world physics within a Director movie, than lingo. Therefore an addition to the planning of this project has been to research the Havok physics engine.

Experiment with various Space warps

After in-depth research it has been found that space warps do not export to a format which Director recognises. Therefore this task has been altered to that of researching the Havok physics engine.

Meet With Dr Newbury

This has been a regular meeting every week to discuss the development of the project and take feedback from Dr Newbury.

2 Gantt chart

3 Email contacts with Forge FX

From: ben smith [mailto:benbo4@hotmail.com]
Sent: Thursday, November 24, 2005 1:28 AM
To: info@forgefx.com
Subject: 3D Mountain formation Simulator

Dear sir/Madam

I am currently in the process of undertaking a final year project at university in England. My project is to create a 3D mountain formation simulator (think tectonic plates), in which the user can dynamically alter different variables within director such as force, mass etc., and a resulting animation showing the tectonic movement will be played, each time a variable is changed, the animation will respond and look different. I am modelling the graphics in 3D Studio max, and creating the interactivity Via Director. I was very impressed with your case study for Prentice Hall.. And i can see that you have used 3DS MAX and director to a similar effect and to some degree of success. My question to you is this:

I am aware that you cannot export deformable meshes into director without using BONES.. I am extremely interested in the techniques you used for your wave formation simulator and your topographic map. Do either of these involve bones? I realise that this project is as much of an investigation into the possibilities of integrating these two software's, as much as it is to create a tutorial. Do you think my project is feasible? I would very much appreciate ANY advise or help you may have time to give me.

Thanks in Advance

Ben Smith.

Hi Ben,

Check out the meshDeform modifier. This allows you to use Lingo script within Director to programmatically move vertices within a given 3D model. This technique does not use bones. In general, it's more feasible to alter the geometry of models that you've built from scratch via code rather than altering models created in Max. If you build the models yourself in code, you know exactly how they're put together, this is how we made the wave simulator. Models that come in from Max may be easier to generate but can also have a more complex structure which is harder to manipulate via code in the end.

For a good source of info from experts, subscribe to:
<http://nuttybar.drama.uga.edu/mailman/listinfo/dir3d-l>

Cheers,


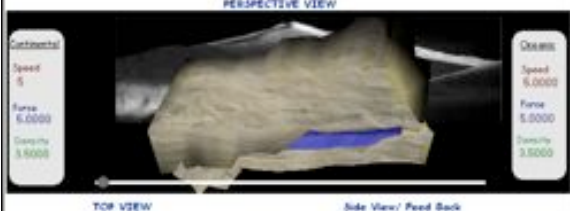
Adam

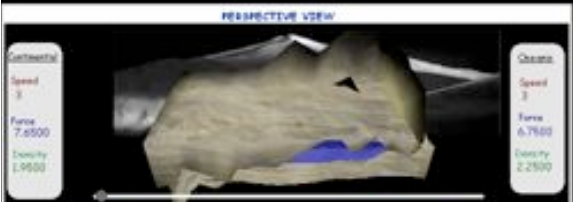
// Adam Kane (akane@forgefx.com)
// Lead Programmer, ForgeFX, LLC
// 808.946.9879 office
// 808.375.2067 cell
// www.forgefx.com

4 Ad-hoc Test

Page	Test instruction	Result	Correct
IntroPage	Check that Skip intro button goes to main menu	Goes to Main menu	✓
	Check intro plays smoothly until the end	Some minor jumps	✗
Main Menu Page	Check Learn link	Goes to Tectonic plates page	✓
	Check Experiment Link	Goes to Simulation Page	✓
	Check Play Link	Opens mountain sculptor window	✓
Tectonic Plates Page	Check Main Menu Link	Goes to Main menu	✓
	Check Next Page Link	Goes to Fold Mountain Page	✓
	Check Back Link	Goes to Main menu	✓
	Ensure Pangea Timeline works	Images fade in and out, according to movement of slider	✓
	Check scroll arrows	Text is scrollable	✓
	Check More about Fold Mountains Link	Opens 3D Diagram Page	✓
	Check Animations link	Goes to Animation Guide	✓
	Check Lets See Some Pictures link	Goes to Picture index page	✓
	Check Take me to the Simulator link	Goes to getting to know the simulator page	✓
	Check Back Link	Goes back to main Menu page	✓
3D Diagram	Rotate	Image appears to rotate, controlled by mouse movement	✓
	Zoomin	Camera zooms in	✓
	Zoomout	Camera zooms out	✓
	Reset	Camera is reset	✓
	Pick Model to change Text	Rendering of clicked layer changes to wire frame, and text is displayed explaining how layer is made	✓
	Close window	Window shuts down	✓
Animation Guide	Converging link	Jumps to converging animation page	✓
	Diverging link	Jumps to diverging animation page	✓
	Transform Link	Jumps to transform animation page	✓
	Back link	Goes back to Tectonic plates page	✓
	Next link	Goes to converging animation page	✓
	Main Menu link	Jumps to main menu	✓
Converging Plates	Rewind button	Animation returns to beginning	✓
	Pause button	Animation pauses	✓
	Play button	Animation plays	✓
	Back button	Goes back to animation guide	✓
	Next button	Goes to Diverging plates page	✓

	Main Menu Button	Goes back to main menu	✓
	Scroll Bar	Text is scrollable	✓
Diverging Plates	Rewind button	Animation returns to beginning	✓
	Pause button	Animation pauses	✓
	Play button	Animation plays	✓
	Back button	Goes back to Converging	✓
	Next button	Goes to Transform animation	✓
	Main Menu button	Goes back to main Menu	✓
	Scroll Bar	Text is scrollable	✓
Transform Plates	Rewind button	Animation returns to beginning	✓
	Pause button	Animation pauses	✓
	Play button	Animation plays	✓
	Back button	Goes to Diverging plate animation page	✓
	Next button	Goes to converging plate animation	✓
	Main Menu button	Goes back to main menu	✓
	Scroll Bar	Text is scrollable	✓
Fold mountain Page	Picture links	Each pictures open the slide show window	✓
	Back button	Goes back to fold mountain page	✓
	Next button	Goes to getting to know the simulator page	✓
	Main Menu Button	Goes to main menu	✓
Picture slideshow Page	Back button	Goes back one pictures	✓
	Next button	Goes forward on pictures	✓
	Off button	Shuts down window	✓
	Fade in	New images fades in smoothly when user clicks next	✓
	Fade Out	Current image fades out smoothly when user clicks next	✓
	Typewriter text effect	Text appears with typewriter effect, and good readable speed	✓
Getting to know the simulator Page	Back button	Goes back to fold mountain page	✓
	Next button	Goes to control panel page	✓
	Main Menu button	Goes back to main menu	✓
	Control Panel link	Goes to control panel page	✓
	Cameras link	Goes to cameras page	✓
	Feedback Link	Goes to Feedback Page	✓
	Text scroll bars	Text is scrollable	✓

Control Panel Page	Back button	Goes back to getting to know simulator page	✓
	Next button	Goes to cameras page	✓
	Main Menu button	Goes back to main menu	✓
	Fade in effect	Steps fade in slowly and are readable	✓
Cameras Page	Text scroll bars	Text is scrollable	✓
	Back button	Goes back to control panel page	✓
	Next button	Goes to feedback page	✓
	Main Menu button	Goes back to main menu	✓
Feedback page	Text scroll bars	Text is scrollable	✓
	Back button	Goes back to cameras page	✓
	Next button	Goes to getting to know simulator page	✓
	Main Menu button	Goes back to main menu	✓
Simulation Page	Click Radio Buttons check that sliders adjust to previous settings, as shown in feedback panels	Radio buttons are selectable, sliders immediately jump to value shown in corresponding feedback.	✓
	Alter sliders and click set values, make sure feedback panel reflects values that have just been set	Sliders are adjustable, clicking set values, automatically enters values into feedback panel	✓
	Set values and click step, ensure plates move and react upon collision	Plates collide and respond as expected	✓
	Click step 4 times, plates should move and deform	Plates translate, and geometry is altered	✓
	Click reset, values should return to zero	Sliders return to zero, but values on feedback panel remain the same	✗
	Select oceanic plate and assign grid texture, basalt texture and granite texture	Textures map on to plate	✓
	Select continental plate and assign grid texture, basalt texture and granite texture	Textures map on to plate	✓
	Set all inputs to Minimum values and click simulate.	 <p>Result is as expected, plates collide slowly, and geometry deformation is minimal</p>	✓
	Set all inputs to Maximum values and click simulate.	 <p>Result is as expected, with plates colliding fast, and geometry deforming a lot.</p>	✓

















	Set all inputs to medium values and click simulate.	 <p>Result is good, collision is medium speed, deformation is more than when values are set to minimum, and less than when values are set to maximum</p>	✓
	Check Go to movie button, movie should play back smoothly, and be rewindable	Animations is saveable, movie can be played back and rewind	✓
	Place mouse over sliders to test mouse over instructions text	Instruction text appears in side view panel	✓
	Check main Menu button	Goes to main menu	✓
Mountain Sculptor Page	Close window button	Window closes	✓
	click on basalt	Texture changes correctly	✓
	Click on granite	Texture changes correctly	✓
	Click on grid	Texture changes correctly	✓
	Click on plate with right mouse button 3 times	Area that was clicked, drops 3 times, on y axis	✓
	Click on plate with left mouse button 3 times	Area that was clicked, raises 3 times, on y axis	✓

5 Open ended user study

6 Specific task user testing

7 Requirement Analysis Checklist

Requirement Analysis	In order to fulfil requirement Program must have :
<p>Create a tutorial explaining how Fold Mountains are formed demonstrating the tectonic activities involved, and the result of these activities.</p> <p>Requirement met 100%</p>	<p>Rendered Animation of fold mountain. ✓</p> <p>Explanation of Pangea and tectonic plates. ✓</p> <p>Pangea timeline. ✓</p> <p>3D still model with interactive controls. ✓</p> <p>Fold Mountain Interactive simulation. ✓</p>
<p>Prompt the user to enter variable values into the program such as: Force(N); Mass; time; rock type.</p> <p>Requirement met 75%</p>	<p>Prompt to tell the user to go back if values are .not entered ✗</p> <p>Help to suggest values to be entered. ✓</p> <p>Visibility status for feedback of current values. ✓</p> <p>Reset value options. ✓</p>
<p>Introduce causality where altering any given variable affects the resultant 3D animation.</p> <p>Requirement met 100%</p>	<p>Ability to alter value once it is entered. ✓</p> <p>Instant reaction showing change in simulation once value is changed. ✓</p> <p>Ability to alter values during a simulation. ✓</p>
<p>Make the program interactive and enjoyable to use.</p> <p>Requirement met 86%</p>	<p>correctly working, interactive Animations. ✓</p> <p>Interactive slide show. ✓</p> <p>Sound samples and effects to enhance interactions. ✗</p> <p>AVI videos. ✓</p> <p>Ability to manipulate 3D model using interactive tools. ✓</p> <p>Buttons working correctly and link to correct Place. ✓</p> <p>a “Play” section with interactive model in which geometry is editable at runtime. ✓</p>
<p>Investigate the possibilities of combining 3DS MAX technologies with that of Macromedia Director</p> <p>Requirement 100% met</p>	<p>Investigate and explanation of Dynamics ✓</p> <p>Investigate and explanation of Havok. ✓</p> <p>Investigate and explanation of Bones. ✓</p> <p>Investigate and explanation of Spacewarps. ✓</p> <p>Investigate and explanation of Shockwave. ✓</p> <p>Investigate and explanation of #MeshDeform. ✓</p>
<p>Program must accord with what is required from National Curriculum.</p> <p>Requirement 100% met</p>	<p>Explain Global distribution of continental plates. Tensional and compressional margins. ✓</p> <p>Explain Characteristic features and formation of fold mountains. ✓</p> <p>Occurrence and measurement of earthquakes. ✓</p>

<p>The user will be able to control the camera view of the animation as it is in motion.</p> <p>The user will be able to zoom in and out of the animation.</p> <p>Requirement 50% met</p>	<p>Camera tool to enable user to move camera around image during simulation. </p> <p>Multiple Camera views of simulation. </p>
<p>The program will have a tutorial on how to use the simulator.</p> <p>Requirement 100% met</p>	<p>Section to explain how to change variables. </p> <p>Section to describe how tutorial works. </p> <p>Section to explain control panel. </p> <p>Section to explain camera views. </p>
<p>The program will also act as a tutorial on other types of tectonic activity.</p> <p>Requirement 100% met</p>	<p>Section to explain and depict Dome mountains. </p> <p>Section to explain and depict Fault block mountains. </p> <p>Section to explain and depict Plateau mountains. </p> <p>Section to explain and depict Volcanic mountains. </p> <p>Animations of other types of mountain formations. </p> <p>Pictures to illustrate different types of mountain formations. </p>
Primary Objectives 90% Met	
Secondary Objectives	
The program will simulate more than one type of tectonic movement.	
The user will be able to ‘play’ with their new mountain range, by creating snow and wind.	
The user will be able to place a character in the scene to explore the terrain.	
The program will have a quiz/game on mountain formations.	
Secondary Objectives 0% Met	

8 System Wide code and behaviours

```
--changes colour of sprite upon mouse over this is used for ALL navigation text controls
on mousewithin me
  sprite(me.spritenum).color = RGB(100,100,100)
end

on mouseleave me
  sprite(me.spritenum).color = RGB(0,0,0)
end

--Makes sure player head does not carry on past this point unless directed by nav action
on exitFrame me
  go to the frame
end
```

Front Menu Page



9 Jump to learn and load Text

--This behaviour Ensures that when the "learn" button is clicked, the player head will jump to the label "learn" and load the tectonics.rtf file into the textbox.

```
on mouseUp me
    go to "learn"
    member(193).filename = "tectonics.rtf"
    member(193).width = 380
    member(193).height = 320
    member(193).boxType = #scroll
    member(193).fontSize = 12
    member(133).text = "Tectonic Plates" -- loads title
end
```

10 Jump to Simulator and load Text

```
on mouseup me
    go to "init"
end
```

11 Jump Mountain Sculptor

```
on mouseup me
    window("refinedpickingFinal4.dir").open()
    window("refinedpickingFinal4.dir").title = "Mountain Sculptor"
end
```

Learn Menu Page



12 Jump to "fold" and load Text

--This behaviour Ensures that when the "next" button is clicked, the player head will jump to the label "fold" and load the foldMountains.rtf file into the text box

```
on mouseup me
    member(193).filename = "fold.rtf"
    member(193).width = 380
    member(193).height = 300
    member(193).fontSize=12
    member(133).text = "Fold Mountains"
    go to "fold"
end
```

13 Jump to Front Menu

-- goes to label "start" when clicked.

```

on mouseup me
    go to "start"
end

```

14 Pangea Timeline Scroll Script

-- This has been modified and added too. The original script was taken from <http://www.ullala.at/experiments/ilwidgets/index.html> reference number[21] only the modified code is shown below.

```

--some code
on getCurrent4 me, val
    if pValue = val then exit
    pValue = min(max(val, pMinValue), pMaxValue)
    pThumbOffset = GetButtonOffset(pValue)
    me.DrawElement()
    sendSprite(me.spriteNum, #ChangeValue, pValue, 1) -- sends current value to handler
    updateStage
end

```

```

end
--some code

```

15 Pangaea Display handler

--This script receives value from scroll bar script, and decides which sprites to make visible and which must not be seen.

```

property p2
property p3
property p4

on changeimage val,val2
    pValue = val2
    p1 = sprite(11)
    p2 = sprite(12)
    p3 = sprite(13)
    p4 = sprite(14)

    if val2 = "2" then
        p1.visible= true
        p1.blend = "100"
        p2.visible= false
        p3.visible= false
        p4.visible= false

    else if val2 = "3" then
        p1.visible= true
        p1.blend = "50"
        p2.visible= true
        p2.blend = "50"
        p3.visible= false
        p4.visible= false

    else if val2 = "4" then
        p1.visible= false
        p2.visible= true
        p2.blend = "100"
        p3.visible= false
        p4.visible= false

    else if val2 = "5" then
        p1.visible= false
        p2.visible= true
        p2.blend = "50"
        p3.visible= true
        p3.blend = "50"
        p4.visible= false

    else if val2 = "6" then
        p1.visible= false
        p2.visible= false
        p3.visible= true
        p3.blend = "100"
        p4.visible= false

    else if val2 = "7" then
        p1.visible= false
        p2.visible= false
        p3.visible= true

```

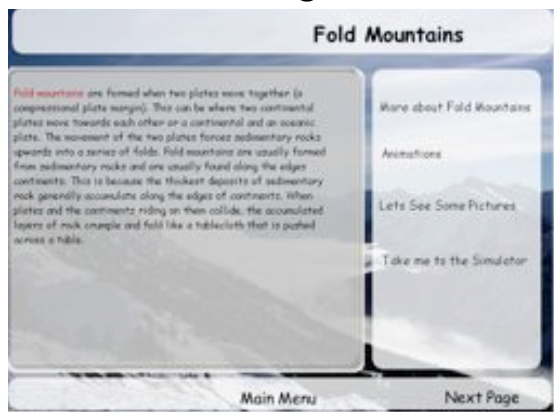
```

p3.blend = "50"
p4.visible= true
p4.blend = "50"

else if val2 = "8" then
  p1.visible= false
  p2.visible= false
  p3.visible= false
  p4.visible= true
  p4.blend = "100"
end if
end

```

Fold Mountain Page



16 Load Diagrams.dir

--This behaviour opens the "diagram.dir" movie in a new window, and specifies a new title for the window. This will run when the user click on "more about fold mountains"

```

on mouseUp me
  window("diagram.dir").open()
  window("diagram.dir").title = "3dDiagram"
end

```

17 Jump to "animations"

--This behaviour ensures that when the "animations" button is clicked, the player head will jump to the label "Animations" and load the animations.rtf file into the text box

```

on mouseUp me
  member(193).filename = "animations.rtf"
  member(133).text = "Animation Guide"
  go to "Animations"
end

```

18 Jump to "Lets see some pictures"

--This behaviour ensures that when the "lets see some pictures" button is clicked, the player head will jump to the label "fold2"

```

on mouseUp me
  go to "fold2"
end

```

19 Jump to tutorial1

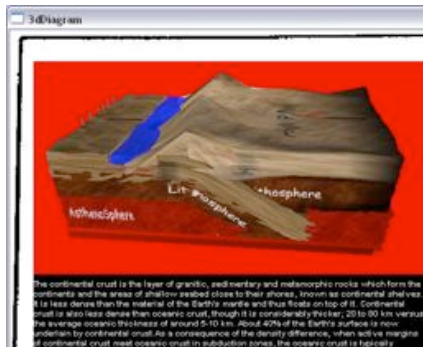
--This behaviour ensures that when the "take me to the simulator" button is clicked, the player head will jump to the label "tutorial1" and load the tutorial.rtf file into the text box

```

on mouseUp me
  member(193).filename = "tutorial1.rtf"
  member(193).height = 100
  member(193).width = 380
  member(193).boxType = #scroll
  member(193).fontSize=12
  member(133).text = "Other Mountain Types"
  go to "tutorial1"
  member(133).text = "Getting to know the Simulator"
end

```

More about fold Mountains



20 Image D, Pick layer, display text, change render style

```
property spritenum, origin

global scenel
property whichmodel
property modellist

on preparemovie
    scenel = member("3d")
    -- In our preparemovie event we will take care of
    -- naming all globals that we will use through the
    -- rest of the movie - generally we will be setting
    -- a global variable with a reference to the SW3D
    -- castmember to ease our typing burden later on
    -- create a new shader
end

on beginsprite
    preparemovie
    origin = point(sprite(spritenum).left, sprite(spritenum).top)
    --create backdrop--
end

on mousedown me
    pt = the clickloc - origin

    -- get detailed info
    modellist = sprite(spritenum).camera.modelsUnderLoc(pt, 1, #detailed)

    --make sure that a model has been hit
    if modellist <> [] then
        --find which model was hit
        whichmodel = modellist[1][#model].name
        modellist[1][#model].shader.renderstyle = #wire
        sendallsprites(#changetext,modellist[1][#model].name , whichmodel)
    end if
end

on mouseup me
    if modellist <> [] then
        --find which model was hit
        whichmodel = modellist[1][#model].name
        modellist[1][#model].shader.renderstyle = #fill
        sendallsprites(#changetext,modellist[1][#model].name , whichmodel)
    end if
end

----- The #changeText method, alters the text box depending which layer has been hit.
on ChangeText models, whichmodel
```



```

if whichmodel = "lithosphere1" then
  member(5).text="Oceanic crust is the part of Earth's lithosphere which underlies the
ocean basins.It is thinner and generally less than 10 kilometres thick, but more dense than
continental crust, with a density of about 3.3 grams per cubic centimeter. Oceanic crust is
composed of mafic basaltic rocks. Most of the present day oceanic crust is less than 200
million years old because it is continuously being created at oceanic ridges and destroyed
by being pulled back into the mantle in subduction zones by the processes of plate
tectonics."
end if

if whichmodel = "lithosphere2" then
  member(5).text ="The continental crust is the layer of granitic, sedimentary and
metamorphic rocks which form the continents and the areas of shallow seabed close to their
shores, known as continental shelves. It is less dense than the material of the Earth's
mantle and thus floats on top of it. Continental crust is also less dense than oceanic
crust, though it is considerably thicker; 20 to 80 km versus the average oceanic thickness
of around 5-10 km. About 40% of the Earth's surface is now underlain by continental
crust.As a consequence of the density difference, when active margins of continental crust
meet oceanic crust in subduction zones, the oceanic crust is typically subducted back into
the mantle. Because of its relative low density, continental crust is only rarely subducted
or re-cycled back into the mantle (for instance, where continental crustal blocks collide
and overthicken, causing deep melting). For this reason the oldest rocks on Earth are
within the cratons or cores of the continents, rather than in repeatedly recycled oceanic
crust; the oldest continental rock is the Acasta Gneiss at 4.01 Ga, while the oldest
oceanic crust is of Jurassic age. The height of mountain ranges is usually related to the
thickness of crust. This results from the isostasy associated with orogeny (mountain
formation). The crust is thickened by the compressive forces related to subduction or
continental collision. The buoyancy of the crust forces it upwards, the forces of the
collisional stress balanced by gravity and erosion. This forms a keel or mountain root
beneath the mountain range, which is where the thickest crust is found. The thinnest
continental crust is found in rift zones, where the crust is thinned by detachment faulting
and eventually severed, replaced by oceanic crust. The edges of continental fragments
formed this way (both sides of the Atlantic Ocean, for example) are termed passive
margins."
end if

if whichModel = "asthenosphere1" then
  member(5).text= "The asthenosphere is the region of the Earth between 100-200 km below
the surface, but perhaps extending as deep as 400 km, that is the weak or soft zone in the
upper mantle. It lies just below the lithosphere, which is involved in plate movements and
isostatic adjustments. In spite of its heat, pressures keep it plastic, and it has a
relatively low density. Seismic waves, the speed of which decrease with the softness of a
medium, pass relatively slowly though the asthenosphere, the cue that originally alerted
seismologists to its presence; thus it has been given the name low-velocity zone. Under
the thin oceanic plates the asthenosphere is usually much nearer the seafloor surface, and
at mid-ocean ridges it rises to within a few kilometres of the ocean floor."
end if

if whichModel = "asthenosphere2" then
  member(5).text= "The asthenosphere is the region of the Earth between 100-200 km below
the surface, but perhaps extending as deep as 400 km, that is the weak or soft zone in the
upper mantle. It lies just below the lithosphere, which is involved in plate movements and
isostatic adjustments. In spite of its heat, pressures keep it plastic, and it has a
relatively low density. Seismic waves, the speed of which decrease with the softness of a
medium, pass relatively slowly though the asthenosphere, the cue that originally alerted
seismologists to its presence; thus it has been given the name low-velocity zone. Under
the thin oceanic plates the asthenosphere is usually much nearer the seafloor surface, and
at mid-ocean ridges it rises to within a few kilometres of the ocean floor."
end if

if whichModel = "ocean" then
  member(5).text ="The nature of a convergent boundary depends on the type of lithosphere in
the plates that are colliding. Where a dense oceanic plate collides with a less-dense
continental plate, the oceanic plate is typically thrust underneath, forming a subduction
zone. At the surface, the topographic expression is commonly an oceanic trench on the ocean
side and a mountain range on the continental side. An example of a continental-oceanic
subduction zone is the area along the western coast of South America where the oceanic
Nazca Plate is being subducted beneath the continental South American Plate. As the
subducting plate descends, its temperature rises driving off volatiles (most importantly
water). As this water rises into the mantle of the overriding plate, it lowers its melting
temperature, resulting in the formation of magma with large amounts of dissolved gases.
This can erupt to the surface, forming long chains of volcanoes inland from the continental
shelf and parallel to it. The continental spine of South America is dense with this type of
volcano. In North America the Cascade mountain range, extending north from California's
Sierra Nevada, is also of this type. Such volcanoes are characterized by alternating
periods of quiet and episodic eruptions that start with explosive gas expulsion with fine
particles of glassy volcanic ash and spongy cinders, followed by a rebuilding phase with
hot magma. The entire Pacific ocean boundary is surrounded by long stretches of volcanoes

```

and is known collectively as The Ring of Fire. Where two continental plates collide the plates either crumple and compress or one plate burrows under or (potentially) overrides the other. Either action will create extensive mountain ranges. The most dramatic effect seen is where the northern margins of the Indian subcontinental plate is being thrust under a portion of the Eurasian plate, lifting it and creating the Himalaya."

```
----- define length width and type of text box
```

```
end if
member(5).width = 480
member(5).height = 150
member(5).boxType = #scroll
member(5).fontSize = 10
end
```

21 Orbit camera

```
----- This is a pre-built director behaviour that has been modified, only the modified parts
of the code is shown here
```

```
--
--some code
--
--PURPOSE: Checks to see if a trigger event has been received. If so it executes
--          the requested handler.
--ACCEPTS: 'aScript' as a reference to a script member.
--RETURNS: Nothing.
```

```
-----
on enterFrame(me)
  if action = "rotate" then -- modification to check action

    if pMemberStateOK then

      if pInitialized then

        if pQueuedTriggeredEvents.count > 0 then

          repeat with j = 1 to pQueuedTriggeredEvents.count
            tEvent = pQueuedTriggeredEvents[j]

            case tEvent of
              #orbitX:
                me.orbitX(pModel)
              #orbitY:
                me.orbitY(pModel)
              #orbitZ:
                me.orbitZ(pModel, #vertical)

              #orbitXY:
                me.orbitX(pModel)
                me.orbitY(pModel)
              #orbitXZ:
                me.orbitX(pModel)
                me.orbitZ(pModel, #vertical)
              #orbitYZ:
                me.orbitY(pModel)
                me.orbitZ(pModel, #horizontal)

            end case
          end repeat

          pQueuedTriggeredEvents = []

        end if

      else
        pInitialized = me.initialize()
      end if

    else
      pMemberStateOK = me.isMemberStateOK()
    end if

  end if
end enterFrame
```

```
-----
--PURPOSE: Saves the current mouse information and calculates the vector under
--          the current mouse location.
--ACCEPTS: 'me' as an instance of this script.
--RETURNS: Nothing.
-----
```

```

on mouseDown(me)

    if action = "rotate" then -- modification to check action

        if pInitialized then
            pLastMouseLocV = the mouseV
            pLastMouseLocH = the mouseH
            pLastMouseWorldVector = pCamera.spriteSpaceToWorldSpace(the mouseLoc)
        end if
    end if
end mouseDown

-----
--PURPOSE: Saves the current mouse information and calculates the vector under
--           the current mouse location.
--ACCEPTS: 'me' as an instance of this script.
--RETURNS: Nothing.
-----
--Public Custom Handlers-----
-----
--PURPOSE: Convert a descriptive, localized string into an associated handler
--           and execute that handler.
--ACCEPTS: 'me' as an instance of this script.
--           'aEvent' as a string.
--RETURNS: Nothing
-----
on registerEvent(me, aEvent, aTriggerSpriteNum, aTargetBehaviorInstance)
    if action = "rotate" then -- modification to check action

        if pInitialized then
            if me = aTargetBehaviorInstance then

                tHandler = \
                [#orbitXY:"Camera orbit on X and Y",\
                 #orbitXZ:"Camera orbit on X and Z",\
                 #orbitYZ:"Camera orbit on Y and Z",\
                 #orbitX: "Camera orbit on X axis",\
                 #orbitY: "Camera orbit on Y axis",\
                 #orbitZ: "Camera orbit on Z axis" ].getOne(aEvent)

                if symbolP(tHandler) then
                    pQueuedTriggeredEvents.add(tHandler)
                end if

            end if
        end if
    end if

end registerEvent
--
--some code
--

```

22 Orbit camera trigger

```

global action
on mouseUp me
    action = "rotate" -- tells system which action to do
    cursor 2 -- sets cursor to crosshairs
end

```

23 Check cursor symbol

```

global action
property pSprite
property pCursor
-- ensures that the cursor always shows the correct symbol
--whilst mouse is over the image according to which action is selected
on mousewithin me
    pSprite = sprite(me.spriteNum)
    if action = "rotate" then
        pCurspr=2
        pSprite.cursor = pCursor
    else if action = "zoomin" then
        pCurspr=302
    else if action = "zoomout" then
        pCurspr=303
    end if
end
end

```

24 Camera zoom in

---- This is a pre-built director behaviour that has been modified, only the modified parts of the code is shown here

```

--
--some code
--

--PURPOSE: Checks to see if a trigger event has been received. If so it executes
--           the requested handler.
--ACCEPTS: 'me' as an instance of this script.
--RETURNS: Nothing.
-----
on enterFrame(me)

    if action = "zoomin" then -- code enhancement
        if pMemberStateOK then

            if pInitialized then

                if pQueuedTriggeredEvents.count > 0 then

                    repeat with j = 1 to pQueuedTriggeredEvents.count
                        tEvent = pQueuedTriggeredEvents[j]

                        case tEvent of
                            #dollyCameraIn:
                                me.dollyCamera(-pDollyAmount)
                            #dollyCameraOut:
                                me.dollyCamera(pDollyAmount)
                        end case

                    end repeat

                    pQueuedTriggeredEvents = []

                end if

            else
                pInitialized = me.initialize()

            end if

        else
            pMemberStateOK = me.isMemberStateOK()
        end if
    end if
end enterFrame
-----
--Public Custom Handlers-----
-----
--PURPOSE: Convert a descriptive, localized string into an associated handler
--           and execute that handler.
--ACCEPTS: 'me' as an instance of this script.
--           'aEvent' as a string.
--RETURNS: Nothing
-----
on registerEvent(me, aEvent, aTriggerSpriteNum, aTargetBehaviorInstance)

    if action="zoomin" then -- code enhancement
        if pInitialized then
            if me = aTargetBehaviorInstance then

                tHandler = [#dollyCameraIn: "Move Camera In", \
                            #dollyCameraOut: "Move Camera Out"].getOne(aEvent)

                if symbolP(tHandler) then
                    pQueuedTriggeredEvents.add(tHandler)
                end if

            end if
        end if
    end if

end registerEvent
--
-- more code

```

25 Zoom camera trigger

```

global action
on mouseUp me
    action = "zoomin"
    cursor 302

```

end

26 Zoom camera out

---- This is a pre-built director behaviour that has been modified, only the modified parts of the code is shown here

```
--
--some code
--
--PURPOSE: Checks to see if a trigger event has been recieved. If so it executes
--          the requested handler.
--ACCEPTS: 'me' as an instance of this script.
--RETURNS: Nothing.
```

```
-----
on enterFrame(me)

  if action = "zoomout" then --check action
    if pMemberStateOK then

      if pInitialized then

        if pQueuedTriggeredEvents.count > 0 then

          repeat with j = 1 to pQueuedTriggeredEvents.count
            tEvent = pQueuedTriggeredEvents[j]

            case tEvent of
              #dollyCameraIn:
                me.dollyCamera(-pDollyAmount)
              #dollyCameraOut:
                me.dollyCamera(pDollyAmount)
            end case

          end repeat

          pQueuedTriggeredEvents = []

        end if

      else

        pInitialized = me.initialize()

      end if

    else

      pMemberStateOK = me.isMemberStateOK()
    end if
  end if
end enterFrame
```

```
-----
--Public Custom Handlers-----
-----
--PURPOSE: Convert a descriptive, localized string into an associated handler
--          and execute that handler.
--ACCEPTS: 'me' as an instance of this script.
--          'aEvent' as a string.
--RETURNS: Nothing
-----
```

```
on registerEvent(me, aEvent, aTriggerSpriteNum, aTargetBehaviorInstance)
```

```
  if action = "zoomout" then --check action

    if pInitialized then
      if me = aTargetBehaviorInstance then

        tHandler = [#dollyCameraIn: "Move Camera In", \
                    #dollyCameraOut: "Move Camera Out"].getOne(aEvent)

        if symbolP(tHandler) then
          pQueuedTriggeredEvents.add(tHandler)
        end if

      end if
    end if
  end if
```

```
end registerEvent
-- more code
```

27 Zoom out camera trigger

```
global action
on mouseUp me
    action = "out"
    cursor 303
end
```

28 Reset Camera

---- This is a pre-built director behaviour that has been modified, only the modified parts of the code is shown here

```
--
--some code
--
on enterFrame(me)

    if action = "reset" then -- check action
        if pMemberStateOK then

            if pInitialized then

                if pQueuedTriggeredEvents.count > 0 then
                    --Although it is possible that multiple triggers could be sent to this
                    --behaviour, due to its nature it doesn't make sense to do so.
                    tEvent = pQueuedTriggeredEvents[1]
                    case tEvent of
                        #resetCamera: me.resetCamera()
                    end case
                    pQueuedTriggeredEvents = []
                    --Purge the queue.
                end if

            else
                pInitialized = me.initialize()
            end if

        else
            pMemberStateOK = me.isMemberStateOK()
        end if
    end if
end enterFrame

--PURPOSE: Convert a descriptive, localized string into an associated handler
--          and execute that handler.
--ACCEPTS: 'me' as an instance of this script.
--          'aEvent' as a string.
--RETURNS: Nothing
-----
```

```
on registerEvent(me, aEvent, aTriggerSpriteNum, aTargetBehaviorInstance)
```

```
    if action = "reset" then
        if pInitialized then
            if me = aTargetBehaviorInstance then

                tHandler = \
[#resetCamera: "Reset camera"].getOne(aEvent)

                if symbolP(tHandler) then
                    pQueuedTriggeredEvents.add(tHandler)
                end if

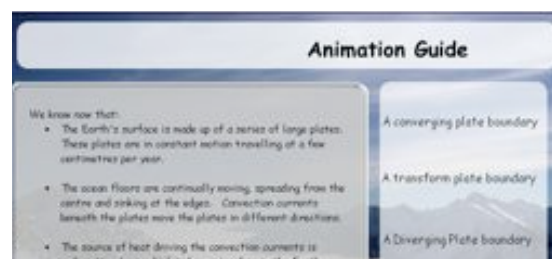
            end if
        end if
    end if
end registerEvent
```

```
--
```

```
--some code
```

29 Reset Camera Trigger

```
global scene
global action
on mouseUp me
    action = "reset" -- set action field
end
```



Animation Menu Page

30 Button Navigations

```
-- Converge
on mouseup me
  go to "animation1"
end
--Transform
on mouseup me
  go to "animation2"
end
--Diverge
on mouseup me
  go to "animation3"
end
--Back
on mouseup me
  go to "fold"
end
--Next
on mouseup me
  go to "animation1"
end
--Main menu
on mouseup me
  go to "start"
end
```

Animation1 Page

31 Button Navigations

```
--Play
on mouseup me
  sprite(4).movieRate = 0.5
end
--Pause
on mouseup me
  sprite(4).movieRate = 0
end
--Rewind
on mouseup me
  sprite(4).movietime = 0
  sprite(4).movieRate = 0
end

--Back
on mouseup me
  go to "animation"
end
--Next
on mouseup me
  go to "animation2"
end
--Mainmenu
on mouseup me
  go to "start"
end
```



Animation2 Page

32 Button Navigations

```
--Play
on mouseup me
  sprite(4).movieRate = 0.5
end
--Pause
on mouseup me
  sprite(4).movieRate = 0
end
--Rewind
on mouseup me
  sprite(4).movietime = 0
  sprite(4).movieRate = 0
end
```



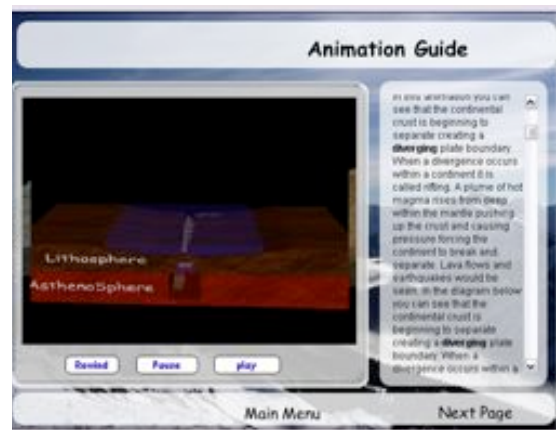
```
end
```

```
--Back
on mouseup me
  go to "animation1"
end
--Next
on mouseup me
  go to "animation3"
end
--Mainmenu
on mouseup me
  go to "start"
end
```

Animation 3 Page 33 Button Navigations

```
--Play
on mouseup me
  sprite(4).movieRate = 0.5
end
--Pause
on mouseup me
  sprite(4).movieRate = 0
end
--Rewind
on mouseup me
  sprite(4).movietime = 0
  sprite(4).movierate = 0
end
```

```
--Back
on mouseup me
  go to "animation"
end
--Next
on mouseup me
  go to "animation1"
end
--Mainmenu
on mouseup me
  go to "start"
end
```



Images and other mountains 34 Button Navigations

```
--Opens picture slide show when image is clicked on
on mouseup me
  window("mountainImages.dir").open()
  window("mountainImages.dir").title = "Pictures"
end
```

```
--Back
on mouseup me
  go to "fold"
end
--Next
on mouseup me
  go to "tutorial1"
end
--Mainmenu
on mouseup me
  go to "start"
end
```



Slide Show

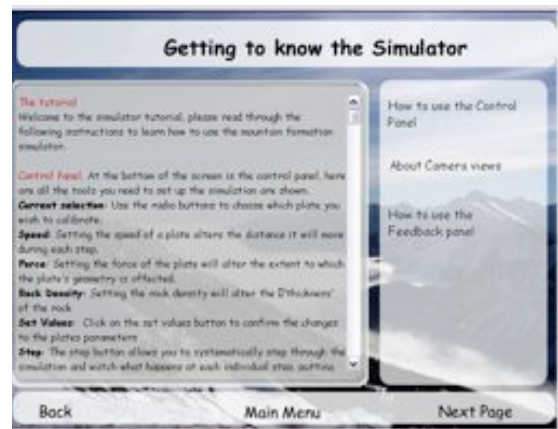
35 Close slideshow

```
-- most of the code on this movie uses built in behaviours. With exception o the off button
below
on mousedown me
  window("mountainImages.dir").close()
end
```

Simulation Tutorial

36 Button Navigations

```
--How to use control panel
on mouseUp me
  go to "control Panel"
end
--Cameras
on mouseUp me
  go to "control Panel"
end
--feedback
on mouseUp me
  go to "control Panel"
end
```



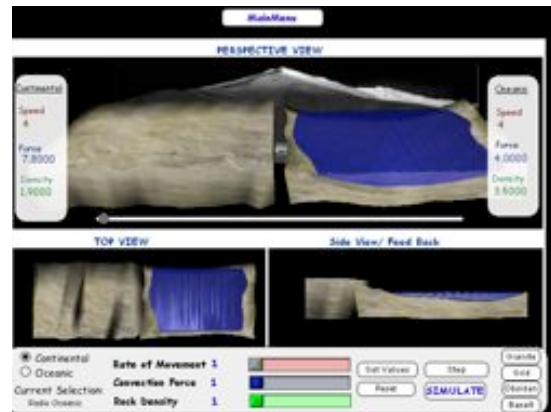
Simulation Page

37 Radio Button

-- This Radio Button behaviour has been modified from the website

<http://www.ullala.at/experiments/ilwidgets/index.html>, Reference [22]

the modifications are stated in bold.



```

-----
-- This behavior controls a group of
-- buttons so that only
-- one button displays a "down" (selected)
-- member while
-- the other buttons display an "up"
-- (deselected) member.
--
-- The button members consist of
-- an up member immediately followed by a down member in
-- the cast.
--
-----

-- pUpMember - The up member of the button
-- pDownMember - The down member of the button
-- pButtonName - The button's name which is defined in the
-- behavior's parameter dialog box. The first word of the
-- name determines the button's group
-- pGroupName - Name of the button's group. The button
-- responds to messages sent by the other members of the
-- group only
property pUpMember, pDownMember, pButtonName, pGroupName, pSelectedPlate

-- This handler determines the up member, down member, and
-- group name for the button.
on beginSprite me

    -- Defines the button's current member
    set currentSprite to the spriteNum of me
    set currentMember to the member of sprite currentSprite
    set currentMemberNumber to the number of member currentMember

    -- Defines the up and down members for the button
    if the name of member currentMember contains "up" then

        -- Sets the up member to the sprite's member when
        -- the sprite initially contains the up member
        set pUpMember to member currentMemberNumber
        set downMemberNumber to currentMemberNumber+1
        set pDownMember to member downMemberNumber

    else

        -- Sets the up member to the member immediately before
        -- the sprite's member in the cast when the sprite
        -- contains the down member (that is, the button is
        -- initially "on")
        set pUpMember to currentMemberNumber-1
        set downMemberNumber to currentMemberNumber
        set pDownMember to member downMemberNumber
        pSelectedPlate = pButtonName
        sendAllSprites(#getSelectedPlate, PSelectedPlate)

    end if

    -- Extracts and defines the button's group name
    if stringP(pButtonName) then
        set pGroupName to word 1 of pButtonName
    end if

end

```

```

-- This handler operates the button when it is clicked.
on mouseDown me

    checkWhich
    remberVal
    set currentSprite to the spriteNum of me

    -- Tells all the buttons in the group to display
    -- their up members
    sendAllSprites(#upButtons, pGroupName)

    -- Displays the down (selected) member
    set the member of sprite currentSprite to member pDownMember

    -- Displays the selected button's name in the text field
    set the text of member "Display Selection" to pButtonName

end

-- This handler responds to messages from other buttons in
-- the group by displaying the button's up member
on upButtons me, groupID

    -- Displays the button's up member when the upButtons
    -- message comes from a button with a matching groupID
    if groupID = pGroupName then
        set the member of sprite the spriteNum of me to pUpMember
    end if

end

-- This handler lets you set the button's group name
-- in the behavior's parameter dialog box.
on getPropertyDescriptionList

    return [ #pButtonName: [ #default: "unnamed", #format: #string, #comment: "Button Name" ] ]
end

-- This handler provides the description in the behavior
-- parameter dialog box.
on getBehaviorDescription
    return "Name of the button. First word defines the button's group."
end

on CheckWhich
    -- this handler checks which plate is selected

    pSelectedPlate = pButtonName
    sendAllSprites(#getSelectedPlate, PSelectedPlate)

end

on remberVal
    --This handler updates the values of the variables, if they have already been set
    if pSelectedPlate = "Radio Oceanic" then

        inputval = value(field "oceanicSpeedField")
        sendallsprites (#getCurrent1,inputval)

        inputval = value(field "oceanicforceField")
        sendallsprites (#getCurrent2,inputval)

        inputval = value(field "oceanicDensity")
        sendallsprites (#getCurrent3,inputval)

    else

        inputval = value(field "continentalSpeedField")
        sendallsprites (#getCurrent1,inputval)

        inputval = value(field "continentalforceField")
        sendallsprites (#getCurrent2,inputval)
    end
end

```

```

        inputval = value(field "continentaldensityField")
        sendallsprites (#getCurrent3,inputval)

    end if
end

```

38 Update rate of movement value

```

global scene
-- update value 1

property pMember
property pSetHandler
property pGetHandler
property pValue

on beginSprite me
    pMember = sprite(me.spriteNum).member
    h = pSetHandler
    put "g" into char 1 of h
    pGetHandler = symbol(h)
    me.updateValue()
end

on UpdateValue me
    val = sendAllSprites(pGetHandler)
    val = integer(val)
    if val <> pValue or voidP(pValue) then
        pValue = val
        put pValue into member pMember
    end if
end

on deactivateElement me
    val = integer(the text of member pMember)
    if integerP(val) then
        pValue = val
        sendAllSprites(pSetHandler, pValue)
        sendAllSprites(#updateValue)
    end if
end

on DisplayValue me, val
    -- only for #SetValue/#GetValue
    if pGetHandler = #GetValue and val <> pValue then
        pValue = val
        put val into member pMember
        sendAllSprites(#getSpeed, val)--This send the value of the field to the handler
responsible for the plate movement
    end if
end

on getPropertyDescriptionList
    descr = [:]
    addProp descr, #pSetHandler, [#format:#symbol, #default:#SetValue, #comment:"handler
name"]
    RETURN descr
end

```

39 Update convection force value

```

-- update value 2
property pMember
property pSetHandler
property pGetHandler
property pValue

on beginSprite me
    pMember = sprite(me.spriteNum).member
    h = pSetHandler
    put "g" into char 1 of h
    pGetHandler = symbol(h)
    me.updateValue()
end

on UpdateValue me
    val = sendAllSprites(pGetHandler)
    val = integer(val) -- just for this demo :-)
```

```

    if val <> pValue or voidP(pValue) then
        pValue = val
        put pValue into member pMember
    end if
end

on deactivateElement me
    val = integer(the text of member pMember)
    if integerP(val) then
        pValue = val
        sendAllSprites(pSetHandler, pValue)
        sendAllSprites(#updateValue)
    end if
end

on DisplayValue2 me, val
    -- only for #SetValue/#GetValue
    if pGetHandler = #GetValue and val <> pValue then
        pValue = val
        put val into member pMember
        sendAllSprites(#getForce, val)
    end if
end

on getPropertyDescriptionList
    descr = [:]
    addProp descr, #pSetHandler, [#format:#symbol, #default:#SetValue, #comment:"handler
name"]
    RETURN descr
end

```

40 Update rock density value

```

-- update value 3

property pMember
property pSetHandler
property pGetHandler
property pValue

on beginSprite me
    pMember = sprite(me.spriteNum).member
    h = pSetHandler
    put "g" into char 1 of h
    pGetHandler = symbol(h)
    me.updateValue()
end

on UpdateValue me
    val = sendAllSprites(pGetHandler)
    val = integer(val) -- just for this demo :-)
    if val <> pValue or voidP(pValue) then
        pValue = val
        put pValue into member pMember
    end if
end

on deactivateElement me
    val = integer(the text of member pMember)
    if integerP(val) then
        pValue = val
        sendAllSprites(pSetHandler, pValue)
        sendAllSprites(#updateValue)
    end if
end

on DisplayValue3 me, val
    -- only for #SetValue/#GetValue
    if pGetHandler = #GetValue and val <> pValue then
        pValue = val
        put val into member pMember
        sendAllSprites(#getDensity, val)
    end if
end

on getPropertyDescriptionList
    descr = [:]

```

```

    addProp descr, #pSetHandler, [#format:#symbol, #default:#SetValue, #comment:"handler
name"]
    RETURN descr
end

```

41 Set rate of Movement slider

-- This has been modified and added too. The original script was taken from <http://www.ullala.at/experiments/ilwidgets/index.html> reference number[21] only the modified code is shown below.

--some code

```

on getCurrent1 me, val--receives value from val, if already set, so that slider can be
adjusted to value
    if pValue = val then exit
    pValue = min(max(val, pMinValue), pMaxValue)
    pThumbOffset = GetButtonOffset(pValue)
    me.DrawElement()
    sendSprite(me.spriteNum, #ChangeValue, pValue, 1)--Send value to Change value handler
    updateStage

```

end

end

```

on SetValue me, val
    if pValue = val then exit--sets slider to the values received
    pValue = min(max(val, pMinValue), pMaxValue)
    pThumbOffset = GetButtonOffset(pValue)
    me.DrawElement()
end

```

--some code

42 Set rate of movement feedback

--sets the feed back field

```

on ChangeValue me, val
    sendAllSprites(#DisplayValue, val)
end

```

43 Set convection force slider

-- This has been modified and added too. The original script was taken from <http://www.ullala.at/experiments/ilwidgets/index.html> reference number[21] only the modified code is shown below.

--some code

```

on getCurrent2 me, val--receives value from val, if already set, so that slider can be
adjusted to value
    if pValue = val then exit
    pValue = min(max(val, pMinValue), pMaxValue)
    pThumbOffset = GetButtonOffset(pValue)
    me.DrawElement()
    sendSprite(me.spriteNum, #ChangeValue, pValue, 1)--Send value to Change value handler
    updateStage

```

end

end

```

on SetValue me, val
    if pValue = val then exit--sets slider to the values received
    pValue = min(max(val, pMinValue), pMaxValue)
    pThumbOffset = GetButtonOffset(pValue)
    me.DrawElement()
end

```

--some code

44 Set convection force feedback

--sets the feed back field

```

on ChangeValue me, val
    sendAllSprites(#DisplayValue2, val)
end

```

45 Set rock density slider

-- This has been modified and added too. The original script was taken from <http://www.ullala.at/experiments/ilwidgets/index.html> reference number[21]

only the modified code is shown below.

--some code

```
on getCurrent3 me, val--receives value from val, if already set, so that slider can be
adjusted to value
  if pValue = val then exit
  pValue = min(max(val, pMinValue), pMaxValue)
  pThumbOffset = GetButtonOffset(pValue)
  me.DrawElement()
  sendSprite(me.spriteNum, #ChangeValue, pValue, 1)--Send value to Change value handler
  updateStage
```

end

end

```
on SetValue me, val
  if pValue = val then exit--sets slider to the values received
  pValue = min(max(val, pMinValue), pMaxValue)
  pThumbOffset = GetButtonOffset(pValue)
  me.DrawElement()
end
```

--some code

46 Set density feedback

--sets the feed back field

```
on ChangeValue me, val
  sendAllSprites(#DisplayValue3, val)
end
```

47 Set values Button

-- check which plate is selected and sets up input values

```
property vertexToGo, v_cnt, myplane
property box1
property box2
property cnt
property cntup
property pMember

property pspeed
property pforce
property pDensity
property pSelectedPlate

property pOceanicspeed
property pOceanicforce
property pOceanicDensity

property pContspeed
property pContforce
property pContDensity

on mousedown me
  pSelectedPlate = word 2 of member("Display Selection").text
  sendallsprites(#setupVariables, pspeed, pdensity,pforce, pselectedplate)
end

--Get speed information from slider
on getSpeed me, speed
  pspeed = speed
end

--Get Force information From slider
on getForce me, forces
  pforce = forces
end

--get Density of Rock
on getDensity me, Densities
  pDensity = Densities
end
```

48 Reset Button

```
-- re-initialises 3d Scene, and re-sets all input values.
global scene
property val

on mousedown
  initialize
  val = 0
  sendallsprites(#getCurrent1, val)
  sendallsprites(#getCurrent2, val)
  sendallsprites(#getCurrent3, val)
  go to "run"

end
```

49 Step Button

```
--most important code for simulation, loads input variables into algorithms, and deals with
collision detection and deforms model geometry and
global scene    -- we will reference the SW3D in this script
property vertexToGo, v_cnt, myplane
property box1
property box2
property cnt
property cntup
property pMember

property pspeed
property pforce
property pDensity
property pSelectedPlate

property pOceanicspeed
property pOceanicforce
property pOceanicDensity

property pContspeed
property pContforce
property pContDensity

--Set up specific variables for plate selection--
on setupVariables(a,inspeed,indensity,inforce,inplate)
  --nothing has been selected, default values are needed

  if inplate ="Oceanic" then

    pOceanicspeed = inspeed
    pOceanicDensity = 4.5 - indensity
    pOceanicforce = inforce * (4.5-indensity)

    if pOceanicforce >10 then
      pOceanicforce = 10
    end if

    if pContDensity < pOceanicDensity then
      alert "Remember that the oceanic plate is always more dense than the continental this
is why is subducts, you must adjust the setting so the continental plate is less dense"
      exit
    end if

    put pOceanicspeed into member("oceanicSpeedField")
    put pOceanicForce into member("oceanicForceField")
    put ((4.5 + indensity) - pOceanicDensity)/2 into member("oceanicDensity")

  else if inplate ="continental" then

    pContspeed = inspeed
    pContDensity = 4.5 - indensity
    pContforce = inforce * (4.5-indensity)

    if pContforce >10 then
      pContforce = 10
    end if

    if pContDensity < pOceanicDensity then
      alert "Remember that the oceanic plate is always more dense than the continental this
is why is subducts, you must adjust the setting so the continental plate is less dense"
      exit
    end if

  end if
```



```

end if

put pContspeed into member("continentalSpeedField")
put pContForce into member("continentalForceField")
put ((4.5 + inDensity) - pContDensity)/2 into member("continentalDensityField")

end if

end

on mousedown(me)

--setupVariables
-- initialise plates
repeat with i = 1 to 4
    if scene.model[i].name = "box02" then
        box2 = scene.model[i]
    end if
    if scene.model[i].name = "box01" then
        box1 = scene.model[i]
    end if
    if scene.model[i].name = "water" then
        water = scene.model[i]
    end if
end repeat

box1.collision.mode = #sphere --Assign bounding sphere for collision detection
box2.collision.mode = #sphere
water.collision.mode = #sphere

box1.translate(-pOceanicspeed,0,0) --load in speed input
box2.translate(pContspeed,0,0)
water.translate(-pOceanicspeed,0,0)

box1.collision.resolve = true ) --Resolve collision
box2.collision.resolve = true
water.collision.resolve = true

box1.collision.resolve = false --Turn off collision detector so plate can move again
box2.collision.resolve = false
water.collision.resolve = false

box2.collision.setcollisioncallback(#myhandler, me) --call handler when collision is
found

end

-- when collision is found

on myhandler(me,data)

    whichmodelmade = data.modelA
    whichmodelcrash = data.modelB
    where = data.pointofcontact
    datanormal = data.collisionNormal

    targetpos = whichmodelmade.getworldtransform().position
    modellist = scene.modelsunderray(vector(where.X, 10000, where.Z), vector(0,-1,0),
#detailed)

    --make sure that a model has been hit
    if modellist <> [] then
        modelsInvolved = modellist.count
        repeat with l = 1 to modelsInvolved

            -----FOR BOX 1 (the oceanic Plate)-----
            if modellist[l][#model].name = "box01" then
                --## getting the face out of the "modellist"
                whichface = modellist[l][#faceID]
                --## getting the list of vertices that belong to the face

```

```

        vertlist =
modellist[1][#model].meshdeform.mesh[modellist[1][#meshID]].face[whichface]

--## storing the complete vertexlist into "vertex_list"
vertex_list=modellist[1][#model].meshDeform.mesh[modellist[1][#meshID]].vertexlist
--## how many entrys are in the vertexlist
cnt=vertex_list.count
--## going through the vertices of the clicked face, allways 3 points for each face
repeat with i = 1 to vertlist.count
    --## set the "vert" to the right entry of the vertexlist
    vert =
modellist[1][#model].meshdeform.mesh[modellist[1][#meshID]].vertexlist[vertlist[i]]

--## select all points from the mesh that are equal to "vert",this can be more than 3.
vertexToGo=[]
repeat with n=1 to vertex_list.count
    if vertex_list[n] = vert then
        vertexToGo.add(n)
    end if
end repeat

--## now all points of the mesh that share the 3 points of the face are selected
--## and count then entrys of the list
v_cnt=vertexToGo.count

--## now setting all points from the vertexToGo list to a different place
repeat with m=1 to v_cnt
    --## storing the old position of the vertex
    oldvert =
modellist[1][#model].meshDeform.mesh[modellist[1][#meshID]].vertexlist[vertexToGo[m]]
    --## calculating some new position
    --## this here will calculate a vector that shows from the pivot of the mesh (centre)
    --## into the direction of the vertex and brings it 2.5 units away from the old position
    newvert = vert + vector(-pOceanicDensity,0,-pOceanicforce)
    --## now set the vertex to the new position
modellist[1][#model].meshdeform.mesh[modellist[1][#meshID]].vertexlist[vertexToGo[m]] =
newvert
end repeat
end repeat
end if

-----FOR BOX 2, (The Continental Plate) -----
if modellist[1][#model].name = "box02" then
    -- make sure it was the other plate that caused the collision with the water--
    if whichmodelcrash.name = "box01" then
        --## getting the face out of the "modellist"
        whichface = modellist[1][#faceID]
        --## getting the list of vertices that belong to the face
        vertlist =
modellist[1][#model].meshdeform.mesh[modellist[1][#meshID]].face[whichface]

        --## storing the complete vertexlist into "vertex_list"

vertex_list=modellist[1][#model].meshDeform.mesh[modellist[1][#meshID]].vertexlist
--## how many entrys are in the vertexlist
cnt=vertex_list.count

--## going through the vertices of the clicked face, allways 3 points for each
face
repeat with i = 1 to vertlist.count
    --## set the "vert" to the right entry of the vertexlist
    vert =
modellist[1][#model].meshdeform.mesh[modellist[1][#meshID]].vertexlist[vertlist[i]]

--## select all points from the mesh that are equal to "vert", this can be more than 3
vertexToGo=[]
repeat with n=1 to vertex_list.count
    if vertex_list[n] = vert then
        vertexToGo.add(n)
    end if
end repeat

--## now all points of the mesh that share the 3 points of the face are selected
--## and count then entrys of the list
v_cnt=vertexToGo.count

--## now setting all points from the vertexToGo list to a different place
repeat with m=1 to v_cnt

```

```

--## storing the old position of the vertex
oldvert =
modellist[1][#model].meshDeform.mesh[modellist[1][#meshID]].vertexlist[vertexToGo[m]]

--## calculating some new position
--## this here will calculate a vector that shows from the pivot of the mesh (center)
--## into the direction of the vertex and brings it 2.5 units away from the old position
newvert = vert + vector(pContDensity,0,pContforce)
--## now set the vertex to the new position

modellist[1][#model].meshdeform.mesh[modellist[1][#meshID]].vertexlist[vertexToGo[m]] =
newvert
    end repeat
    end repeat
    end if
    end if

-- -----WATER-----
if modellist[1][#model].name = "water" then
--## getting the face out of the "modellist"
whichface = modellist[1][#faceID]
--## getting the list of vertices that belong to the face
vertlist =
modellist[1][#model].meshdeform.mesh[modellist[1][#meshID]].face[whichface]

--## storing the complete vertexlist into "vertex_list"
vertex_list=modellist[1][#model].meshDeform.mesh[modellist[1][#meshID]].vertexlist
--## how many entrys are in the vertexlist
cnt=vertex_list.count

--## going through the vertices of the clicked face, allways 3 points for each face
repeat with i = 1 to vertlist.count
--## set the "vert" to the right entry of the vertexlist
vert =
modellist[1][#model].meshdeform.mesh[modellist[1][#meshID]].vertexlist[vertlist[i]]

--## select all points from the mesh that are equal to "vert", this can be more than 3
vertexToGo=[]
repeat with n=1 to vertex_list.count
    if vertex_list[n] = vert then
        vertexToGo.add(n)
        -- deleteVertex(member("3dworld"), n)
        -- modellist[1][#model].deletevertex(n) -- doesn't work yet

    end if
end repeat

--## now all points of the mesh that share the 3 points of the face are selected
--## and count then entrys of the list
v_cnt=vertexToGo.count
    end repeat
    end if
    end repeat
    end if
end if
end

```

50 Simulate Button

```

-- Jumps player head, to frame called 'Sim'
on mouseUp me
    deletecast -- delete old images
    go "sim"
end

```

51 'Sim' Frame

```

global scene
on exitFrame me
    repeat with x=1 to 20
        go "sim" --stay on this frame
        scene.model[2].translate(2,0,0) -- move model
        scene.model[3].translate(-2,0,0) -- move model
        scene.model[4].translate(-2,0,0) -- move model
    end repeat
end

```

```

    sprite(56).loc = sprite(56).loc +point(20,0)
    makeBitmap -- create a screen shot
end repeat
go to "endSim"
end

```

52 Making a screen shot of simulation

```

on makeBitmap

    NewBitmap = new(#bitmap, castLib "StagePictures")--saves bitmap in new movie cast library
    NewBitmap.picture = the stage.picture

    if the keyPressed = RETURN then
        castLib( "StagePictures").save()
        halt
    end if

    if the timer > 180 then
        castLib( "StagePictures").save()--prevents deadlock
    end if

end

-- erases previous images, so the can be re-written
on deletecast
    finish = 21
    repeat with i = 2 to finish
        erase member i of castLib "StagePictures"
    end repeat
end on deletecast

```

53 Change texture to Basalt

```

global scene-- we will reference the SW3D in this script
property PSelectedPlate

on mouseDown
    pSelectedPlate = word 2 of member("Display Selection").text --Check which plate is
    selected
    ChangeShaderBasult
end

end

on ChangeShaderBasult
    repeat with i = 1 to 4
        if scene.model[i].name = "box02" then
            obj2 = scene.model[i]
        end if

        if scene.model[i].name = "Box01" then
            obj1 = scene.model[i]
        end if
    end repeat

    ---set Oceanic plate--
    if PSelectedPlate = "Oceanic" then

        shd = scene.shader[7]
        obj1.shaderlist = shd
        end if

    --Set Continental Plate--
    if PSelectedPlate = "continental" then
        shd = scene.shader[7]
        obj2.shaderlist = shd
        end if
    end
end

```

54 change texture to Grid

```

global scene-- we will reference the SW3D in this script
property PSelectedPlate

on mouseDown
    pSelectedPlate = word 2 of member("Display Selection").text --Check which plate is
selected
    ChangeShaderBasult
end

end

on ChangeShaderBasult
repeat with i = 1 to 4
if scene.model[i].name = "box02" then
obj2 = scene.model[i]
end if

if scene.model[i].name = "Box01" then
obj1 = scene.model[i]
end if
end repeat

---set Oceanic plate--
if PSelectedPlate = "Oceanic" then

shd = scene.shader[8]
obj1.shaderlist = shd
end if

--Set Continental Plate--
if PSelectedPlate = "continental" then

shd = scene.shader[8]
obj2.shaderlist = shd
end if
end
end

```

55 change texture to obsidian

```

global scene-- we will reference the SW3D in this script
property PSelectedPlate

on mouseDown
    pSelectedPlate = word 2 of member("Display Selection").text --Check which plate is
selected
    ChangeShaderBasult
end

end

on ChangeShaderBasult
repeat with i = 1 to 4
if scene.model[i].name = "box02" then
obj2 = scene.model[i]
end if

if scene.model[i].name = "Box01" then
obj1 = scene.model[i]
end if
end repeat

---set Oceanic plate--
if PSelectedPlate = "Oceanic" then

shd = scene.shader[4]
obj1.shaderlist = shd
end if

--Set Continental Plate--
if PSelectedPlate = "continental" then

shd = scene.shader[4]
obj2.shaderlist = shd
end if
end
end

```

```
end
```

56 change texture to Basalt

```
global scene-- we will reference the SW3D in this script
property PSelectedPlate

on mouseDown
    pSelectedPlate = word 2 of member("Display Selection").text --Check which plate is
selected
    ChangeShaderBasalt
end

end

on ChangeShaderBasalt
repeat with i = 1 to 4
if scene.model[i].name = "box02" then
obj2 = scene.model[i]
end if

if scene.model[i].name = "Box01" then
obj1 = scene.model[i]
end if
end repeat

---set Oceanic plate--
if PSelectedPlate = "Oceanic" then

shd = scene.shader[6]
obj1.shaderlist = shd
end if

--Set Continental Plate--
if PSelectedPlate = "continental" then

shd = scene.shader[6]
obj2.shaderlist = shd
end if
end
end
```

57 Initialisation script

```
--set's up camera, 3D images, Textures, Shaders, and modifiers

global scene    -- we will reference the SW3D in this script
property init

on initialize
    clearworld(scene) -- call the custom handler clearworld to reset the world to its
default values

    -- initialise plates
    repeat with i = 1 to 4
        if scene.model[i].name = "box02" then
            obj1 = scene.model[i]
        end if
        if scene.model[i].name = "Box01" then
            obj2 = scene.model[i]
        end if
        if scene.model[i].name = "water" then
            obj3 = scene.model[i]
        end if
    end repeat

    --create a new shader
    shd1 = scene.newshader("gridshader1", #standard)
    shd2 = scene.newshader("gridshader2", #standard)
    shd3 = scene.newshader("gridshader3", #standard)

    --create 1st new texture
    txt1 = scene.newtexture("gridtexture1", #fromcastmember, member("basalt"))
```

```

txt1.nearfiltering = false
txt1.quality = #low
txt1.renderformat = #rgba4444-- to optimise performance we allow render format not to
store alpha information
--apply the texture to shader1
shd1.texture = txt1
--create 2nd new texture
txt2 = scene.newtexture("gridtexture2", #fromcastmember, member("granite"))
txt2.nearfiltering = false
txt2.quality = #low
txt2.renderformat = #rgba4444
--apply the texture to shader2
shd2.texture = txt2

--create 3rd new texture
txt3 = scene.newtexture("gridtexture3", #fromcastmember, member("grid"))
txt3.nearfiltering = false
txt3.quality = #low
txt3.renderformat = #rgba4444
shd3.renderstyle = #wire
--apply the texture to shader3
shd3.texture = txt3

--create backdrop--
bg = scene.newtexture("bgimg", #fromcastmember, member("mountic"))

-----Set up water motion---
Waves = scene.motion[1].name
obj3.addmodifier(#keyframeplayer)
obj3.keyframeplayer.play(Waves)

-----box1-----
-- add the collisiondetect modifier
obj1.addmodifier(#collision)

-- add the mesh deform modifier
obj1.addmodifier(#meshdeform)

-- -----box2-----
-- add the collisiondetect modifier
obj2.addmodifier(#collision)

-- add the mesh deform modifier
obj2.addmodifier(#meshdeform)

-- WATER-----
-- add the collisiondetect modifier
obj3.addmodifier(#collision)

-- add the mesh deform modifier
obj3.addmodifier(#meshdeform)

obj1.translate(0,50,0)
obj2.translate(0,50,0)
obj3.translate(0,50,0)

-----Level of detail needs to be concidered for performance issues----
obj1.addmodifier(#lod)
obj2.addmodifier(#lod)
obj3.addmodifier(#lod)

obj1.lod.bias =10
obj2.lod.bias =10
obj3.lod.bias =10

----- camera1-----
scene.camera[1].transform.identity()
scene.model("box01").addchild(scene.camera[1], #preserveparent)
scene.camera[1].fieldofview = 50

scene.camera[1].translate(-5,0,150,#parent)
scene.camera[1].rotate(50,-0,0,#parent)
scene.camera[1].clone("dummy")
scene.camera[1].parent = scene.group("world")
scene.camera[1].pointat(scene.model("sphere01"))
scene.camera[1].addbackdrop(bg, point(100,0),0)--add the backdrop
-----camera 2-----
cam2 = scene.newcamera("top")

```

```

cam2.fieldofview = 70
cam2.transform.identity()
scene.model("box01").addchild(cam2, #preserveparent)
cam2.translate(0,0,100)
cam2.pointat(scene.model("box01"))

-----camera3-----
cam3 = scene.newcamera("sideview")
cam3.fieldofview = 15
cam3.transform.identity()
scene.model("box01").addchild(cam3, #preserveparent)
cam3.translate(0,-20,300)
cam3.rotate(85,0,0,#parent)
cam3.pointat(scene.model("box01"))

sprite(2).addcamera(cam2)
sprite(2).addcamera(cam3)
rr = sprite(1).rect
scene.camera[3].rect = rect(5,200,390,350)
scene.camera[4].rect = rect(320,200,680,350)

end

```

58 Play head script to call initialisation

```

on exitFrame me
    initialize    --call the initialize custom handler
    go to "run"   -- after initialization, go to "run"
end

```

59 Check 3DScene

```

global scene          -- we are going to reference the 3D castmember at least once in this
script

on exitFrame me
    if check3Dready(scene) then
        -- call custom handler: is the SW3D castmember ready?
        go "init"      -- if it is ready, go to the "init" marker
    else
        go the frame    -- if it is not ready, wait here
    end if
end

-- There are two handlers in this script:

-- check3Dready      called with: check3Dready(member("which3Dcastmember"))
-- returns true if the SW3D state is ready (downloaded & ready to operate)

-- clearworld        called with: clearworld(member("which3Dcastmember"))
-- resets the world to its default values & returns true

-----
-- check3Dready is designed to be called either from a repeat loop
-- or at a single instance. It was created to be able to return
-- whether a SW3D castmember's state = 4 (ready), if the SW3D is
-- in any other state, you should not perform operations on it (script error if you do)

on check3Dready whichSW3D    -- takes one argument in the form of
member("which3Dcastmember")
    if whichSW3D.state = 4 then -- a state of 4 tells you the media is ready
        return true           -- if it is ready, return true (for error checking &
debugging)
    else
        return false          --if it isn't ready return false (for error checking &
debugging)
    end if
end

-----
-- clearworld is designed to be used on SW3D castmembers that you want to

```



```

-- contain no data. The resetworld() function resets any 3D castmember
-- to its defaults: 1 camera, 2 lights, 1 modelresource, 1 shader, 1 texture

on clearworld whichSW3D -- takes one argument in the form of
member("which3Dcastmember")
  if check3Dready(whichSW3D) then -- checking to make sure the SW3D is ready
    whichSW3D.resetworld() -- if everything is ready, reset the world to its default
  values
  return true -- return true (for error checking & debugging)
  else
    return false -- the SW3D was not ready, so return false (for error
checking & debugging)
  end if
end

```

60 Naming movie globals

```

-- In our preparemovie event we will take care of
-- naming all globals that we will use through the
-- rest of the movie
global scene

on preparemovie
  scene = member("3dworld")
end

```

61 Mouse over help text

```

on mousewithin me
  sprite(59).visible= true
end
on mouseleave me
  sprite(59).visible= false
end

```

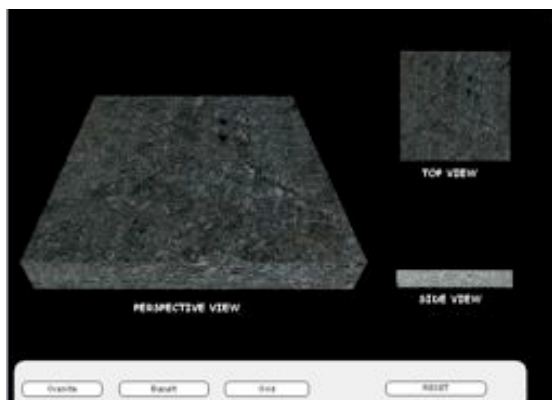
62 Play animation

```

on exitFrame me
  -- this is on 20 frames, to produce 20 images, creating an animation
  set the member of sprite 2 to member 2 of castLib "stagePictures"
  delay 0.2 * 60
end

```

Mountain Sculptor Page



63 Deforming the Plate

```

property spritenum, origin
global scene
on beginsprite

```

```

    origin = point(sprite(spritenum).left, sprite(spritenum).top)
on mousedown
    pt = the clickloc - origin

    -- get detailed info
    modellist = sprite(spritenum).camera.modelsUnderLoc(pt, 1, #detailed)

    --make sure that a model has been hit
    if modellist <> [] then
        -- find out which face was hit
        whichface = modellist[1][#faceID]
        -- find out the 3 vertices that face uses
        vertlist = scene.model[1].meshdeform.mesh[2].face[whichface]

        cnt = scene.model[1].meshdeform.mesh.count
        meshnum = modellist[1].meshID

        --cycle through the 3 vertices
        repeat with x = 1 to 3
            --find the exact vertex location
            vert = scene.model[1].meshdeform.mesh[meshnum].vertexlist[vertlist[x]]
            -- add a small amount to that location
            newvert = vert + vector(0,0,5)
            -- set the vertex to the modified position
            scene.model[1].meshdeform.mesh[meshnum].vertexlist[vertlist[x]] = newvert
        end repeat
    end if
end

on rightmousedown
    pt = the clickloc - origin

    -- get detailed info
    modellist = sprite(spritenum).camera.modelsUnderLoc(pt, 1, #detailed)

    --make sure that a model has been hit
    if modellist <> [] then
        -- find out which face was hit
        whichface = modellist[1][#faceID]
        -- find out the 3 vertices that face uses
        vertlist = scene.model[1].meshdeform.mesh[2].face[whichface]

        cnt = scene.model[1].meshdeform.mesh.count
        meshnum = modellist[1].meshID

        --cycle through the 3 vertices
        repeat with x = 1 to 3
            --find the exact vertex location
            vert = scene.model[1].meshdeform.mesh[meshnum].vertexlist[vertlist[x]]
            -- add a small amount to that location
            newvert = vert + vector(0,0,-5)
            -- set the vertex to the modified position
            scene.model[1].meshdeform.mesh[meshnum].vertexlist[vertlist[x]] = newvert
        end repeat
    end if
end

```

64 Initializing scene

```

global scene    -- we will reference the SW3D in this script

on initialize
    clearworld(scene) -- call the custom handler clearworld to reset the world to its
    default values
    --create a basic plane
    obj = createBOX("myplane", 10,70,70,rgb(100,100,100))

    --increase the length and width vertices
    -- so that there is more to click on
    obj.resource.lengthvertices = 20
    obj.resource.widthvertices = 20
    obj.resource.heightvertices = 20

```

```

--create a new shader
shd1 = scene.newshader("gridshader1", #standard)
shd2 = scene.newshader("gridshader2", #standard)
shd3 = scene.newshader("gridshader3", #standard)

--create 1st new texture
txt1 = scene.newtexture("gridtexture1", #fromcastmember, member("basalt"))
txt1.nearfiltering = false
txt1.quality = #low
--apply the texture to shader1
shd1.texture = txt1

--create 2nd new texture
txt2 = scene.newtexture("gridtexture2", #fromcastmember, member("granite"))
txt2.nearfiltering = false
txt2.quality = #low
--apply the texture to shader2
shd2.texture = txt2

--create 3rd new texture
txt3 = scene.newtexture("gridtexture3", #fromcastmember, member("grid"))
txt3.nearfiltering = false
txt3.quality = #low
--apply the texture to shader3
shd3.texture = txt3

-- apply the shader to all sides of the plane
obj.shaderlist = shd1

-- add the mesh deform modifier
obj.addmodifier(#meshdeform)

--orient the plane
obj.rotate(-50,0,0)

----- camera1-----
scene.camera[1].transform.identity()
scene.model("myplane").addchild(scene.camera[1], #preserveparent)
scene.camera[1].fieldofview = 55

scene.camera[1].translate(0,0,100,#parent)

scene.camera[1].clone("dummy")
scene.camera[1].parent = scene.group("world")

-----camera 2-----
cam = scene.newcamera("topview")
cam.fieldofview = 30
cam.transform.identity()
scene.model("myplane").addchild(cam, #preserveparent)
cam.translate(0,0,180)
cam.pointat(scene.model("myplane"))

-----camera3-----
cam3 = scene.newcamera("sideview")
cam3.fieldofview = 30
cam3.transform.identity()
scene.model("myplane").addchild(cam, #preserveparent)
cam3.translate(0,-165,200)
cam3.pointat(scene.model("myplane"))

sprite(1).addcamera(cam)
sprite(1).addcamera(cam3)
rr = sprite(1).rect
scene.camera[3].rect = rect(600,50,800,220)
scene.camera[4].rect = rect(600,220,800,450)

end

```

65 Create Box and Plane Primitive with lingo

```

-- This code creates a box using pure lingo, by setting the width height and length, and
then assigning a shader and texture to the box
On createbox boxName, L,W,H, boxColor
  If check3Dready(scene) then
    If scene.model(boxname) = void then

```

```

        Res = Scene.newmodelresource(boxName & "res", #box)
        Res.length = L
        Res.width = W
        Res.height = H
        Obj = scene.newmodel(boxName, res)
        Shd = scene.newshader(boxName & "shd", #standard)
        Shd.diffuse = boxColor
        Shd.texture = void
        Obj.shaderlist = shd
        Return scene.model(boxname)
    Else
        Return -3002
    End if
Else
    Return -3001
End if
End
--
-- This code crates a plane with pure lingo, using length and width settings, it is called
in the initialisation script of the 3D scene
On createplane planeName, L, W, planeColor
    If check3Dready(scene) then
        If scene.model(planename) = void then
            Res = Scene.newmodelresource(planeName & "res", #plane)
            Res.length = L
            Res.width = W
            Obj = scene.newmodel(planeName, res)
            Shd = scene.newshader(planeName & "shd", #standard)
            Shd.diffuse = planeColor
            Shd.texture = void
            Obj.shaderlist = shd
            Return scene.model(planename)
        Else
            Return -1002
        End if
    Else
        Return -1001
    End if
End

```

66 Changing textures, granite basalt, and grid

```

on mouseDown me
    ChangeShaderBack
end
global scene    -- we will reference the SW3D in this script
on ChangeShaderBack
    obj = scene.model[1]
    shd = scene.shader[3]
    obj.shaderlist =shd
end
-----
on mouseDown me
    ChangeShader
end

global scene    -- we will reference the SW3D in this script

on ChangeShader
    obj2 = scene.model[1]
    shd2 = scene.shader[4]
    obj2.shaderlist =shd2
end
-----
on mouseDown me
    ChangeShaderGranite
end

global scene    -- we will reference the SW3D in this script

on ChangeGranite
    obj = scene.model[1]
    shd = scene.shader[3]
    obj.shaderlist =shd
end

```

67 Set up globals for movie

```
-- In our preparemovie event we will take care of
-- naming all globals that we will use through the
-- rest of the movie - generally we will be setting
-- a global variable with a reference to the SW3D
-- castmember to ease our typing burden later on
```

```
global scene

on preparemovie
    scene = member("3dworld")
end
```

68 Check scene is ready

-- check3Dready is designed to be called either from a single instance. It was created to be able to return whether a SW3D castmember's state = 4 (ready), if the SW3D is in any other state, do not perform operations on it (script error if you do)

```
on check3Dready whichSW3D      -- takes one argument in the form of
member("which3Dcastmember")
    if whichSW3D.state = 4 then -- a state of 4 tells you the media is ready
        return true           -- if it is ready, return true (for error checking &
debugging)
    else
        return false          --if it isn't ready return false (for error checking &
debugging)
    end if
end

-- clearworld is designed to be used on SW3D castmembers that you want to
-- contain no data. The resetworld() function resets any 3D castmember
-- to its defaults: 1 camera, 2 lights, 1 modelresource, 1 shader, 1 texture

on clearworld whichSW3D      -- takes one argument in the form of
member("which3Dcastmember")
    if check3Dready(whichSW3D) then -- just some last minute double checking to make sure
the SW3D is ready
        whichSW3D.resetworld() -- if everything is ready, reset the world to its default
values
        return true            -- return true (for error checking & debugging)
    else
        return false          -- the SW3D was not ready, so return false (for error
checking & debugging)
    end if
end
```

69 Reset button

```
on mousedown
    initialize -- calls initialization script
end
```

70 Targeted User study