Diary

24/12/12 *(~3 hours)*

- Over the Christmas break, I have decided that I want to do some sort of 3D animation for my Extended Project. The reasons for this are outlined in my rationale.
- I spent today learning about and experimenting with 3D modelling in Blender.
  Although I made some interesting models and effects, I decided that making 3D models in an art package such as Blender was not the way I wanted to approach this project.
- I need to investigate ways of rendering 3D using a computer program. Find out about "OpenGL".

1/1/13 *(~3 hours)*

- I have been researching the way I can get 3D things to display on the screen. I learnt about what OpenGL was, and how to use it. Mostly I used McKesson's tutorial, which I found to be very useful and thorough.
- I implemented a "player" program. This would be a program that sets everything up ready for the actual 3D demo to use, and provides a window for the 3D demo to draw into.
- I made two basic OpenGL examples to get to grips with using it.
  My first test, called `test_triangle`, simply draws a white triangle.
  The second test, called `test_points`, draws some random white points on the screen. I got both tests working on Linux.
- I need to explore OpenGL shaders, and GL shader language (GLSL).

Sources used:

- Learning Modern 3D Graphics Programming
  Jason L. McKesson
  http://arcsynthesis.org/gltut/
  - Written by a reputable graphics programmer.
  - Referenced from Stack Overflow a lot, as a great starting point for learning OpenGL.
  - I am going to use this source a lot, as a reference for OpenGL.
  - Task: get used to OpenGL by writing a program that draws a simple shape on the screen.
- pouët.net
  http://www.pouet.net
  - The biggest online resource for demos. Has lists of demos, groups and a (poorly moderated) forum.
  - I wanted to find out about some of the most famous demos. I also found information about demogroups, and links to YouTube videos (I can't actually run many of the demos on my underpowered laptop...)
  - Questions raised: is it feasible to do *any* of the effects shown on the demos in a project of mine?

8/1/13 *(~1 hour)*

- Now that the Christmas break is over, I wanted to start researching the techniques that I will use.

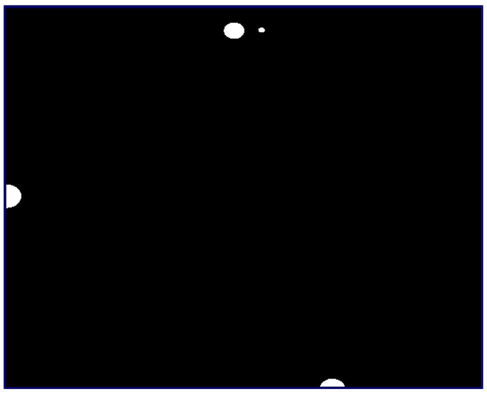
- I was learning about mesh representations and operations, from ryg's blog post.
- I started my research on fluid dynamics and simulation techniques.
  I found out that there are three approaches I can take: solving Navier-Stokes (NS) equations directly, using Smoothed Particle Hydrodynamics (SPH), or Lattice Boltzmann (LBM).
  This demo, numb res, uses SPH. (However, this demo was made to run on machines far powerful than I have access to)
  Blender uses LBM. (However, Blender's fluid simulation is intended for non-realtime rendering)

Sources used:

- "numb res."
  Smash -- direct to video blog
  http://directtovideo.wordpress.com/2011/05/03/numb-res/
    - smash is a democoder from the group Fairlight.
    - Referenced from a pouet.net page about Fairlight's demo "numb res".
    - Stated that fluid simulation using particles is expensive, and even with state-of-the-art optimisations it's not going to work very well on my laptop, unless I use a small number of particles.
    - Questions raised: Just how feasible is it? investigate the signed distance fields

12/1/13 *(~1 hour)*

- I read about using signed distance fields in a presentation at GDC 2012, a presentation at NVScene by iq and a PDF.
- I made a very basic signed distance field test, `test_sdf`, using OpenGL shaders. However, this doesn't display the distance field in 3D, but instead a slice through a 2D plane.



Sources used:

- "Advanced Procedural Rendering in DirectX 11"
  Smash -- GDC 2012
  http://directtovideo.files.wordpress.com/2012/03/gdc_2012_released.pdf
    - Presentation smash gave at the Game Developers Conference. Talked mostly about how he used signed distance fields in his demo "uncovering static". Answered many of my questions about signed distance fields and why they are useful.
    - Questions raised: signed distance fields require ray marching?
- "Rendering Worlds With Two Triangles"
  iq -- NVScene 2008
  http://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf
    - iq is a democoder.
    - This is a presentation he gave at a graphics convention held by NVidia. He talked about various ways of drawing scenes without using polygons, which is the traditional way. This included brief descriptions of signed distance fields and ray marching which agreed with the information I read in smash's presentation. It partially answered my question, however there was not enough detail about ray marching.
    - Questions raised: more details? ray marching sounds expensive -- can I do it on my laptop?
- "Ray Marching Distance Fields in Real-time on WebGL"
  Prutsdom Jiarathanakul
  http://www.seas.upenn.edu/~pjia/raymarch/report.pdf
    - References the presentation iq gave at NVScene.
    - A paper about a computer graphics project done by a student at UPenn. It involves ray marching. The paper goes into great detail about how the ray marching was done, and screenshots of results. It was informational and the results looked good.
    - Questions raised: not sure if this is the right approach for me. Need to investigate polygon based meshes?

13/1/13 *(~2 hours)*

- I successfully compiled and tested my program on Windows.
- I started using git to keep track of older versions of my code and my documents.
- I started implementing a mesh representation based on [ryg's blog post]ryg halfedge-redux. This will be used for "solid" (i.e. non-fluid) stuff, such as the landscape.

Sources used:

- "Half-edges redux"
  ryg -- The ryg blog
  http://fgiesen.wordpress.com/2012/04/03/half-edges-redux/
    - ryg is a well-known democoder from the group farbrausch. His blog post should be trustworthy.
    - This blog post is part of the series "Debris: opening the box", written to explain the technical details behind the famous demo "Debris" -- http://fgiesen.wordpress.com/2012/02/13/debris-opening-the-box/.
      Link referenced from pouet.net.
    - Talked about polygon meshes, specifically the half edge representation of it, and how this particular representation is useful. This sounds much more relevant to me than signed distance fields.
    - Task: make this work in my code
- "Mesh data structures" Denis Zorin, New York University
  http://mrl.nyu.edu/~dzorin/ig04/lecture24/meshes.pdf
    - Explained the different types of mesh representation.
    - I found this source to try to support and confirm the information I found in the blog post above. I found information on a lot of different mesh representations, and the lecturer agrees that the half-edge representation has an advantage as it leads to the cleanest code (with least "if" conditionals).

14/1/13 *(~1 hour)*

- I read further about Navier-Stokes equations. However, I still need to find a source which explains the equations in a way that I can understand, and which is useful to me.
- I wrote my rationale, and expanded my progress log with references.
- I started my timeline plan.

Sources used:

- "Navier-Stokes equations" Computational Fluid Dynamics online wiki http://www.cfd-online.com/Wiki/Navier-Stokes_equations
    - I wanted to find out about what the Navier-Stokes equations calculated, and how. More importantly I wanted to find out, if I implemented the equation, whether it would run quickly enough on my computer.
    - This website seems reliable. Although it is a wiki, giving it inherent potential to have been edited in misleading ways, it is maintained by "CFD online", an online centre for all computational fluid dynamics.
    - The article phrases things in a mathematical way -- not very useful to me, as I only want to assess the feasibility of calculating this on a computer.

15/1/13 *(~30 minutes)*

- Found detailed explanation of Navier-Stokes in a PDF.

Sources used:

- "Fluid Flow for the Rest of Us: Tutorial of the Marker and Cell Method in Computer Graphics"
  David Cline, David Cardon, Parris K. Egbert
  http://people.sc.fsu.edu/~jburkardt/pdf/fluid_flow_for_the_rest_of_us.pdf
    - More accessible. Explains things more thoroughly, and with a computer simulation

    context in mind. Unfortunately this article does not cover the question about the efficiency of the method either.
    - Questions raised: same. I still need to evaluate how feasible it would be to make this work on my laptop.

18/1/13 *(~30 minutes)*

- Reformatted Rationale, Diary and Timeline as Markdown documents. This makes version control of these documents much easier.

19/1/13 *(~3 hours)*

- Continued work on mesh representation. I am trying to implement the vertex split operation using ryg's half-edge blog post.

Sources used:

- "Half-edge based mesh representations: practice"
  ryg -- the ryg blog http://fgiesen.wordpress.com/2012/03/24/half-edge-based-mesh-representations-practice/
    - Explained the vertex split operation, which is required to manipulate meshes.
    - The operation that is described in this blog post is also described similarly in other sources (such as the "Mesh data structures" lecture slides from NYU); however this is the most thorough description that I can find -- other sources do not go into any great detail
    - Tasks: write code that does this and see if it works

20/1/13 *(~2 hours)*

- Continued work on mesh vertex split. Added `test_mesh` to test mesh operations.
  I am finding it quite tricky to code the operations correctly from descriptions of them. I have a poor visual memory so I cannot visualise what each operation does to a mesh, especially in 3D. To work around this, I am having to draw out a lot of small diagrams to help me think about all of the special cases.

21/1/13 *(~4 hours)*

- Expanded on my Rationale and Timeline.
- Continued work on mesh vertex split operation using ryg's half-edge blog post.

22/1/13 *(~4 hours)*

- After more work, the mesh vertex splitting operation seems to work now.
  I still need to test the corner cases.
- I am now trying to implement the face split operation using ryg's half-edge blog post.

23/1/13 *(30 minutes)*

- Added some information to my Rationale.
- Updated my Diary with the amount of time I spent working on my Extended Project each day.

I gave approximate times to the days before today, from memory, as best I can. From today onwards I will track the time I spend more closely.

- I started looking at how to go about implementing a scene graph. I am using information from McKesson's OpenGL tutorial.

Sources used:

- Learning Modern 3D Graphics Programming (chapter 17)
  Jason L. McKesson
  http://www.arcsynthesis.org/gltut/Texturing/Tutorial%2017.html#d0e15853
    - Part of a source I have already previously used (and described).
    - I learnt about how to implement scene graphs using trees.
    - Tasks: digest this and see how it could fit in to my graphics engine.
- "What is a scene graph?" OpenSceneGraph
  http://www.openscenegraph.org/index.php/documentation/knowledge-base/36-what-is-a-scene-graph
    - Written by the author of a reputable and widely used scene graph library. This information should be reliable.
    - I wanted to find information about whether the code examples found in McKesson's tutorial was accurate.
    - I found out about scene graphs, but at an overview level. There were no concrete examples nor details, apart from the very basics. However the information that was present completely agreed with the tutorial.
    - I did manage to find out more about why scene graphs were efficient and popular.

24/1/13 *(2 hours)*

- I changed the style of the code slightly, to allow the use of cleaner OO syntax. This means that future code will be easier to write and understand.
- I have finished implementing the face split operation; it seems to work now.

25/1/13 *(2 hours)*

- I started implementing mesh rendering, which actually draws the mesh to screen. This way I can visually evaluate the results of my code. Up until now I have had to use textual logging when checking of the state of the mesh, which is time-consuming and prone to error.
  I have needed to refer to the documentation on the OpenGL API a lot, which is used for rendering. Most of the information I needed was found in McKesson's tutorial and the OpenGL wiki.
  I managed to successfully draw the mesh primitives that I have so far, a 2D n-sided polygon, with `test_mesh`.
- I sent out an email to ryg, the author of the half-edge mesh representation blog series that I have been using, to ask whether he could give me some information about extrusion. This topic was on his list of blog posts to write, but it seems that he hasn't gotten round to it yet. Hopefully I'll get a response, but if not I will look elsewhere for information.

Sources used:

- Learning Modern 3D Graphics Programming (Chapter 4) Jason L. McKesson
  http://www.arcsynthesis.org/gltut/Positioning/Tutorial%2004.html
    - I am using this to learn about how to use OpenGL to display static polygons.

- OpenGL core api OpenGL.org wiki
  http://www.opengl.org/wiki/Category:Core_API_Reference
    - This is a wiki, and so nothing can be said about reliability. However, the code that I have written based on the documentation found here seems to work, and the wiki is hosted and maintained by OpenGL.org, the official website for OpenGL.
    - I used this site as a reference guide.

27/1/13 *(30 minutes)*

- I am moving all the code which prepared the mesh for rendering, to be associated with a Program (GPU shader) object rather than a Mesh object, as this makes more sense.

28/1/13 *(1 hour)*

- I continued refactoring code.
- I started looking at implementing vector and matrix operations, for transformations such as rotation and scaling.

Sources used:

- AQA FP4 textbook
    - This is a reliable reference for implementing vector and matrix operations.
    - The operations that I am looking at implementing correspond to those that I have been learning about in maths lessons.

29/1/13 *(20 minutes)*

- I read about using quaternions to represent 3D rotations through a cprogramming.com tutorial.

Sources used:

- "Using Quaternion to Perform 3D rotations" confuted, cprogramming.com
  http://www.cprogramming.com/tutorial/3d/quaternions.html
    - This is an overview of how to use quaternions to do 3D rotation. It has a lot of maths, not all of which I completely understand yet. It says that quaternions are like complex numbers, but with three different complex units. This agrees with the other references on quaternions that I have read about previously (on ryg's blog http://fgiesen.wordpress.com/2013/01/07/small-note-on-quaternion-distance-metrics/ and on Wikipedia https://en.wikipedia.org/wiki/Quaternion).
    - I am not sure that this would be the best source to implement code from, however it is a good introduction.
    - Tasks: find another source like this and write the code

30/1/13 *(3 hours)*

- I started writing the vector and matrix code. I am referring to my FP4 maths class notes, and also for implementation details I am gaining inspiration by reading the code from an existing vector/matrix library called GLM.
- Whilst writing the code, I encountered some problems...
```
../include/types.h:244:48: error: could not convert
‘{{(((float)(&(& a)->Mat2::operator[](0ul))-
>Vec2::operator[](0ul)) * ((float)(&(& b)->Mat2::operator[]
(0ul))->Vec2::operator[](0ul))) + (((float)(&(& a)-
>Mat2::operator[](1ul))->Vec2::operator[](0ul)) * ((float)
(&(& b)->Mat2::operator[](0ul))->Vec2::operator[](1ul)))),
((((float)(&(& a)->Mat2::operator[](0ul))->Vec2::operator[]
(1ul)) * ((float)(&(& b)->Mat2::operator[](0ul))-
>Vec2::operator[](0ul))) + (((float)(&(& a)->Mat2::operator[]
(1ul))->Vec2::operator[](1ul)) * ((float)(&(& b)-
>Mat2::operator[](0ul))->Vec2::operator[](1ul))))},
{((((float)(&(& a)->Mat2::operator[](0ul))->Vec2::operator[]
(0ul)) * ((float)(&(& b)->Mat2::operator[](1ul))-
>Vec2::operator[](0ul))) + (((float)(&(& a)->Mat2::operator[]
(1ul))->Vec2::operator[](0ul)) * ((float)(&(& b)-
>Mat2::operator[](1ul))->Vec2::operator[](1ul)))), ((((float)
(&(& a)->Mat2::operator[](0ul))->Vec2::operator[](1ul)) *
((float)(&(& b)->Mat2::operator[](1ul))->Vec2::operator[]
(0ul))) + (((float)(&(& a)->Mat2::operator[](1ul))-
>Vec2::operator[](1ul)) * ((float)(&(& b)->Mat2::operator[]
(1ul))->Vec2::operator[](1ul))))}}' from '<brace-enclosed
initializer list>' to 'Mat2'
```

Fixing the error was rather trivial, once I realised that the best thing to do was not to bother figuring out the error message!

Sources used:

- The GLM library http://glm.g-truc.net/
  - This is a C++ library that contains the code for many vector and matrix operations.
  - It has been referenced from McKesson's tutorial as a very good, high-quality library.
  - The code is openly available for me to examine. I have done this and it has made a lot of things clearer about the syntax of code that I need to write.

31/1/13 *(~30 minutes)*

- Some minor tweaks to the mesh ring generator, and split a header file to speed up compilation.

3/2/13 *(30 minutes)*

- No work done this weekend, because of a mock test tomorrow.
- Previously I have been concerned that I did not have powerful enough hardware on which to develop my demo. That has been somewhat alleviated by some comments by navis on the fact that he developed Lifeforce (a very impressive and large-scale demo) on weaker hardware than mine, by reducing the rendering resolution. However, hopefully I will be able to find (borrow?) some hardware to present the finished demo in higher quality.
- On the other hand, I am also becoming concerned about simulating my waterfall. If I were to use a full particle simulation of the fluid, I would need to simulate each of very many particles. However, particle simulation in realtime is very hard especially with the amount of particles I would need, according to smash, who made an entire demo revolving around fluid simulation on high-end machines. Furthermore, unlike smash, I do not have the hardware to do anything extremely sophisticated.
  Even if I do get a particle simulation working, rendering the particles and making it look like a fluid would be a challenge all in itself. Looking at the "state of the art" in actually

rendering these particles, the techniques either look unrealistic or don't run in real time. The only way that I can envision the waterfall simulation succeeding is by cheating a lot. This means not strictly obeying physics and using "common sense physics" and graphical effects to trick the viewer. This is something I will have to seriously consider this month. If it does not show signs of plausibility soon, I may have to replace the water simulation idea.

Sources used:

- "2007 and now" navis -- Iconoclash blog http://navis-asd.blogspot.co.uk/2010/04/2007-and-now.html
  - This source was written by navis, an influential democoder from the group ASD, who wrote the demo "Lifeforce".
  - I found this source linked through pouet.net.
  - I wanted to find out about the making of such a large demo.
  - In the blog post navis outlines the differences between developing a demo back in 2007, compared to in 2010. He describes how in the past he worked on a slow computer, which meant that he had to run the demo at a very low resolution. However, his demo still looked amazing when run on a fast enough computer at full resolution. This gives me hope that I could develop a convincing physics simulation on my slow laptop, whilst running the final program on a faster, more capable computer.
- "numb res."
  Smash -- direct to video blog
  http://directtovideo.wordpress.com/2011/05/03/numb-res/
- "Surface Reconstruction of SPH Fluids" R. C. Hoetzlein
  http://www.rchoetzlein.com/theory/2010/surface-reconstruction-of-sph-fluids/
  - Written by the author of FLUIDS, a fast open-source fluid simulator
  - States that it is not only hard to simulate fluids using particles, but that it is subsequently very hard to turn those particles into a smooth fluid surface that looks realistic. The blog talks about the state-of-the-art in "reconstructing the surface" from a bunch of particles; it turns out that this is very hard to do in real time.

19/2/13 *(30 minutes)*

- I am very concerned about the lack of progress over the past two weeks.
  The past week (the half-term break) was spent mostly in bed, as I have been rather ill. This has annoyingly meant that I have not been able to make use of the plenty of time that the break provided me with.
  I hope that this break will reinstill enthusiasm in my extended project over the next few weeks, as I have a lot to do in order to reach my target this month.
- I have decided that I am going to write an end-of-month review every month, to summarise the work that I have done in that month, evaluate whether my targets have been reached, and look forwards into the future to see what I am aiming to achieve next month.
  I am starting to write the January report today. (I think this is OK, seeing as I haven't done much in February so far...)
- From the advice that we've been getting in the weekly extended project periods, it is becoming clear that I need to take more careful note of which sources I am using, and what I am gathering from each of them. In order to achieve this I have gone back in my diary and written up the sources that I have been using in some detail, and I will continue to do this, in addition to writing in-depth evaluations for my most important sources.
- On a more positive note, I have started to decide how fluid simulation is going to work in

my demo.
I would like to simulate two things: a waterfall in the side of a cliff, and a vast ocean below, stretching as far as the eye (or camera) can see.
Originally I was thinking of simulating everything with millions of "particles," which represent small units of water that interact with each other to create the effects that we expect such as ripples and waves. However, this is obviously not feasible for the ocean, which would require billions of particles.
Now I am thinking of separating the waterfall and the ocean. The waterfall will still be particle-based, however the ocean will instead be a big 2D plane, and I will keep track of the water level at lots of points on that plane. In this way I can simulate wave behaviour using the concepts that we are learning in physics, such as Huygen's principle, and I will not have to deal with billions of particles. This way, the ocean will be easier to render as well.
I will explain my thoughts in a more detailed write-up separately.

22/2/13 *(10 minutes)*

- I found an explanation of how to perform a wave simulation in a two-dimensional grid at gamedev. I also found a Java applet by Paul Falstad which simulates waves.

Sources used:

- "The Water Effect Explained" Roy Willemse http://www.gamedev.net/page/resources/_/technical/graphics-programming-and-theory/the-water-effect-explained-r915
    - This is a good introduction to wave simulation on a 2D surface, using area sampling on a height map. It even goes into details about how to perform the subsequent raytracing in order to get the diffraction effect on a transparent liquid.
- "Ripple tank" Paul Falstad http://falstad.com/ripple/
    - This is a working Java simulation of waves inside a tank of water. The source code is kindly provided so I am reading through to figure out how the simulation is performed.
    - The source is reliable because the final result clearly works.
    - I am encouraged by the potential of this method, as I have run the Java simulation on my laptop and it works reasonably fast. The graphics and lighting are not 3D, however I can use the technique in the gamedev article to get reasonably fast 3D effects.
    - Task: try implementing the core wave simulation

4/3/13 *(1 hour)*

- I continued writing my January review. I also finished my February review. (Chronology? What's that?)

6/3/13 *(2 hours)*

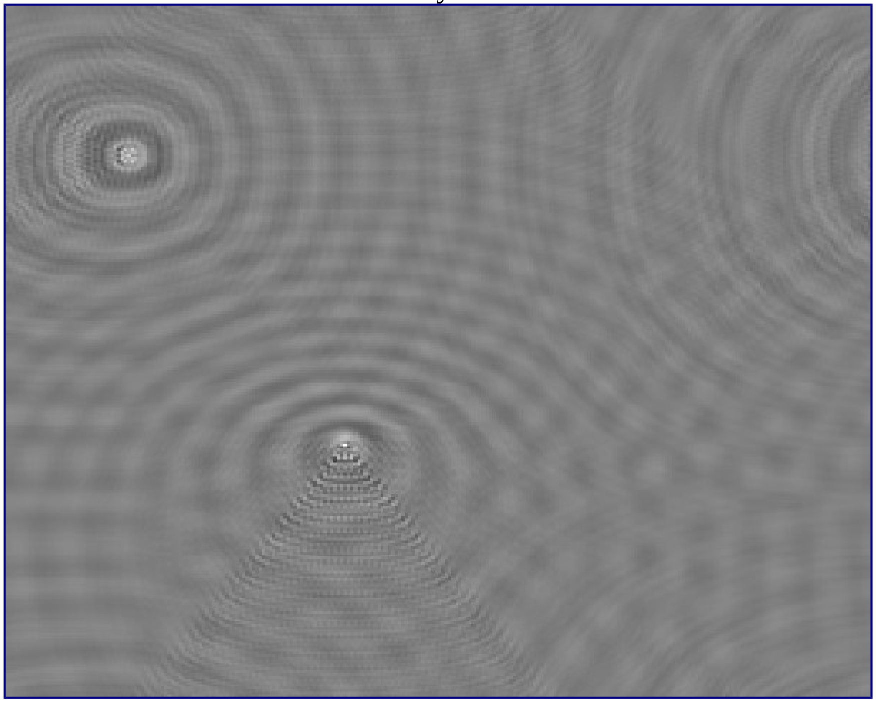
- I worked on the wave simulation.

7/3/13 *(3 hours)*

- After some fiddling, I have a working wave simulation.
  However, it is not perfect. I am simulating in a finite area, and when a ripple reaches the

edge of the simulation it bounces back and continues rippling, whereas I would like it to "disappear" into infinity. The source that I am using to implement the wave simulation does not have this problem; I need to figure out how to stop this. (This does not happen in Falstad's simulation, however the core code that I am using is almost the same as his code?)

- I sent out an email to Paul Falstad with some queries about the mathematical basis of his simulation code (the concepts of which I am using), and asking about the reflection issue.
- I have been reading about using textures in OpenGL, so that I can see the results of my wave simulation. I have successfully written the texture code, and also wrote a scene called `test_wave` that tests all of this.
  The texture code will come in very useful later as well.



29/3/13 *(3 hours)*

- I have been researching on and off for the past few weeks. Unfortunately I have not noted these times in my diary.
- I have fixed edges so that they no longer reflect waves, by damping.
- I am working on improving wave simulation performance. For a 256x256 grid, simulation has improved from 60 to 90 fps.

30/3/13 *(1 hour)*

- More work on wave simulation (100 fps attained).

31/3/13 *(1 hour)*

- More work on wave simulation

20/4/13

- I've been reading lots of online papers on fluid simulation.
- Researched cloth simulation using verlet integration.

Found implementation in JavaScript at this website.

Sources used:

- verlet-js Sub Protocol http://subprotocol.com/verlet-js/
    - Learnt that a physics engine can be based on Verlet integration
    - Has an implemention of Verlet integration in JavaScript. The examples work very effectively.
    - Questions: find out what can and can't be simulated using this physics simulation method. How does it work?

27/4/13

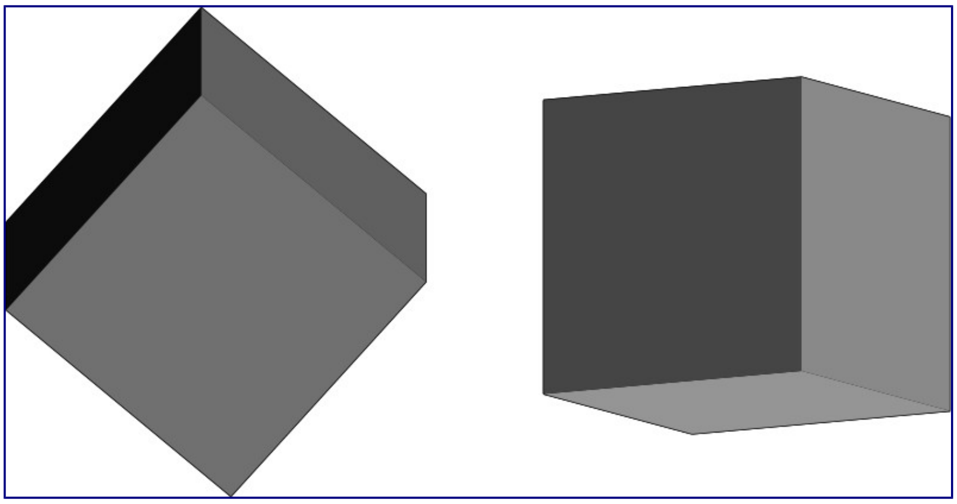
- Looked at efficient collision detection with octrees.

Sources used:

- "Spatial Partitioning and Search Operations with Octrees" Point Cloud Library http://pointclouds.org/documentation/tutorials/octree.php
    - This library is used a lot in graphics and data processing applications, where high performance is needed.
    - The article was based on coding using the PCL, which I am not interested in doing. However, it introduced several concepts, such as k-nearest-neighbour searching, which I will have to use for collision detection between particles.
    - Questions: is it worth implementing octrees or is naive collision testing fast enough for my purposes?

28/4/13

- After a review of progress so far, I am scaling down the ambition of my project to just two things that I want to demonstrate: cloth simulation using verlet integration, and wave simulation using discrete grid-based methods.
- I have updated the Timeline to reflect this.
- I have written an end of April review.
- I have started exploring 3D transformations, lighting and shading using vectors and matrices. I have written a quick prototype demonstrating these three things using JavaScript. The program displays a rotating cube that experiences flat shading from a single light source at infinity. This can be found under `experiments/cube.html`.



29/4/13

- I'm looking at rendering to a texture using an FBO. The documents that I am using is this website.

Sources used:

- "OpenGL Frame Buffer Object 101" "Rob", gamedev.net http://www.gamedev.net/page/resources/_/technical/opengl/opengl-frame-buffer-object-101-r2331
    - This tutorial explains how to use Frame Buffer Objects.
    - I don't necessarily trust online tutorials such as this, however the information I found in it matches that which I found in the OpenGL API reference. In addition it gives me additional information about exactly what parameters I have to give OpenGL in order for the FBO to be "complete".
    - To do: implement in code

30/4/13

- Updated Timeline with more granular targets and estimates of time that it will take.
- Updated and finished end of April review.

6/5/13 *(1 hour)*

- Abstracted shader construction from compilation

7/5/13 *(2 hours)*

- Started working on 3D grid mesh.

8/5/13 *(2 hours)*

- Continued working on 3D grid mesh.
- Fixed vector determinant calculation, and implemented transpose.

31/5/13 *(3 hours)*

- Worked on FBOs
- Not much time for progress -- exams

11/6/13 *(1 hour)*

- Researched text rendering in OpenGL. Used a SDL tutorial and the official SDL-ttf docs.
- Read paper by Valve on smooth text rendering using SDFs.

Sources used:

- "SDL_ttf docs" http://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf.html
    - The official documentation on the SDL_ttf project, which is a library that allows me to render fonts to bitmaps.
- "SDL TTF: Fonts, and how to use them" Alexander Bock http://www.sdltutorials.com/sdl-ttf
    - This is a tutorial about using the SDL_ttf library. It covers the same points as the official documentation, but in a more accessible way, and with illustrations. I checked the code examples relevant to my project against the official documentation, and they matched.
- "What is state-of-the-art for text rendering in OpenGL as of version 4.1?" Stack Overflow http://stackoverflow.com/questions/5262951/what-is-state-of-the-art-for-text-rendering-in-opengl-as-of-version-4-1/5278471#5278471
    - Stack Overflow is a programmer's question and answer website, that has a very good community and reputation.
    - The accepted answer has been "upvoted" 76 times, which means that quite a few programmers have peer-reviewed the answer and approved of it. This increases the amount of trust I have in the answer.
    - I successfully found out about different ways I could render text using OpenGL.
- Improved Alpha-Tested Magnification for Vector Textures and Special Effects Chris Green, Valve http://www.valvesoftware.com/publications/2007/SIGGRAPH2007_AlphaTestedMagnification.pdf
    - This is a reliable source as it was written by a graphics coder at Valve, a large games company. The paper itself has been referenced in many places, including from the Stack Overflow answer mentioned above.
    - I found out about a shader algorithm that could render high quality text using very little processing power.

12/6/13 *(1 hour)*

- Started implementing text rendering.

17/6/13 *(2 hours)*

- Moved to using GLM for vector and matrices. This is a well-tested library and is used in many projects already.
- Found a demonstration of the Verlet integration scheme for simulating physics on a blog, codeflow.

- Updated Timeline

Sources used:

- "Hard Constraints, Easy Solutions" http://codeflow.org/entries/2010/sep/01/hard-constraints-easy-solutions/
    - This is a JavaScript demonstration of Verlet integration. I wanted to find a live demonstration of length constraints, to see the capabilities of Verlet integration. This page provides this, and convinced me that Verlet integration is the way to go.
    - The code examples are not exactly clear -- I need to look these up elsewhere.

19/6/13 *(30 minutes)*

- Fixed a bug in mesh code.

25/6/13 *(1 hour)*

- Continued working on mesh grid code.
- Found a web page as a reference for the Verlet algorithm.
- Started work on Verlet integration physics engine.

Sources used:

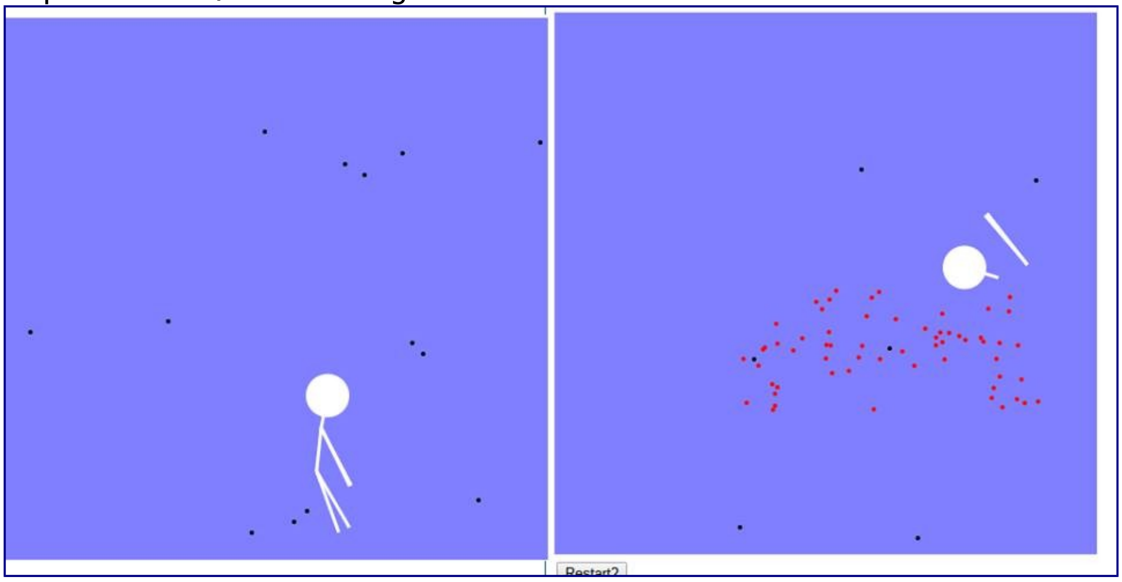
- "The basic Verlet algorithm" http://www.compsoc.man.ac.uk/~lucky/Democritus/Theory/verlet.html#verlet
    - I hoped to find out about the mathematic theory behind the Verlet algorithm so that I could better implement it.
    - I found the equation that forms the fundamentals of it. This agrees with the equation which was implemented in the codeflow JavaScript example.
    - This source does not talk about constraints, which are needed along with the Verlet integration in my physics engine.
- Advanced Character Physics. Thomas Jakobsen http://web.archive.org/web/20080410171619/http://www.teknikus.dk/tj/gdc2001.htm
    - Jakobsen was one of the first to use Verlet integration in a commercial game. This source is considered to have caused a revolution in computer physics simulations back in 2001 when it was published.
    - The algorithm is introduced and agrees with the other two sources so far. In addition he goes into great detail about different kinds of constraints, like length constraints, collision constraints, etc. This is very useful for me.
    - Task: write small prototype and test

28/6/13 *(3 hours)*

- In order to test the algorithm, I wrote a prototype verlet integration physics engine in JavaScript. It moves one end of a ribbon around, leaving the other end free to interact with physics. This experiment can be found under `experiments/verlet.html`.

30/6/13 *(3 hours)*

- In order to further test the Verlet algorithm, I wrote a program to simulate a ragdoll in 2D, which can be controlled by the mouse. In addition, there is collision detection using a technique found in a book (Graphics Gems) that uses maths which I learnt in FP4. I am very pleased with the outcome. This experiment can be found under `experiments/verletrag.html`.



- Started writing May & June end-of-month review.

Sources used:

- Graphics Gems Andrew Glassner
    - This is quite an old book, but still relatively useful, and well regarded in computer graphics circles. It is often referred to, and in this case I found this algorithm referenced in a Stack Overflow answer.
    - Glassner gives how to find intersections in 3D space. I wanted to find intersections in 2D space, but I found it quite easy to specialise the algorithm for this purpose.

23/7/13 *(4 hours)*

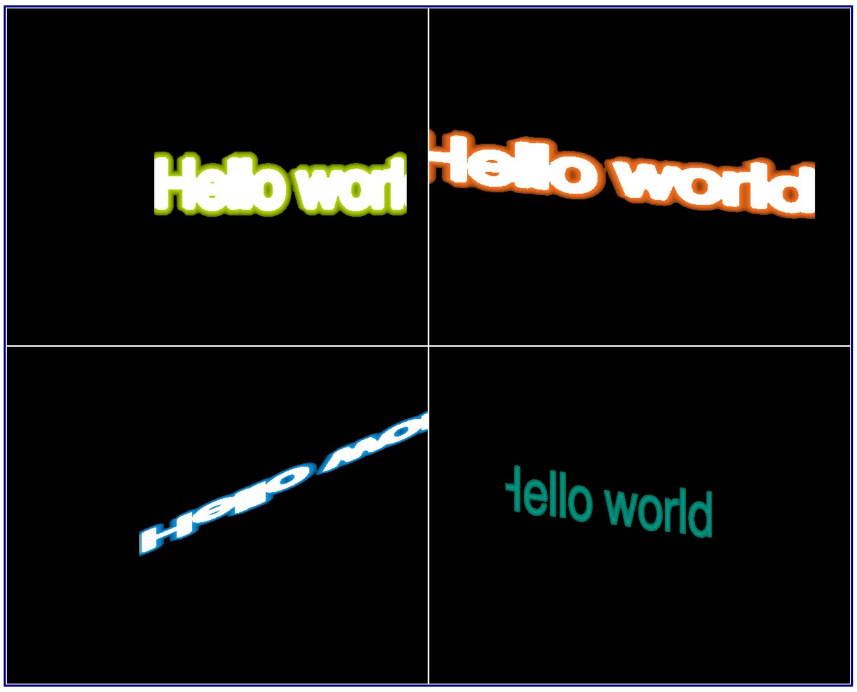
- Arrived home from summer school; work on EP restarts from today.
- Worked on OpenGL shader program and texture code. Not quite working yet.
- Looking to implement text output by tomorrow.

24/7/13 *(4 hours)*

- Continued working on Texture code. Got texture code to work well, by using information from the OpenGL wiki.
- Text output is almost done, except for a bug in the texture upload, which causes the texture to be the wrong width. After an hour's worth of debugging I believe that this is caused by the graphics driver expecting a texture width that is divisible by 8, which I am not providing. I will have to change the code tomorrow to deal with this issue.

25/7/13 *(5 hours)*

- Tracked down the bug where the texture would not render properly if its width was not divisible by 8. This was caused due to the stride not being equal to the number of pixels; there was padding on the end of each row to ensure that each row was aligned to 32 bytes. Fixed this by changing pixel access in SDL_Surfaces to `y*surf->pitch + x` rather than `y*surf->w + x`. Now all sizes of SDL_Surface work.
- Completed the text rendering code. An almost complete implementation of the method outlined in the Valve paper.
- Wrote a test to show off features of the text rendering:
    - High quality output
    - Small texture size (128x128)
    - Customisable boldness
    - Glow effect

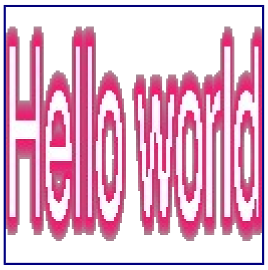


27/7/13 *(4 hours)*

- Worked on PNG writer code. This will allow me to save a texture that is stored in memory into the hard drive for use later.
- Worked on implementing the framebuffer object (FBO). This will allow me not only to render to the screen, but also to an in-memory texture, which I can subsequently save using my PNG writing code, or use in a subsequent mesh.
- Wrote a test `test_rendertex` to test rendering text into a texture using a framebuffer object, and subsequently saving that texture using the PNG writer code. Currently the code is not working: random garbage gets stored in the texture.

28/7/13 *(4 hours)*

- Implemented colour (and transparent) textures using vec4 to represent the RGBA values.
- Fixed and finished both the PNG writer code and the framebuffer code. The test works. Below shows the text as rendered into a square 128x128 texture.



13/08/13 *(4 hours)*

- Implemented matrix stacks. The idea was based on this guide in the OpenGL red book. However I implemented it manually rather than using the OpenGL fixed functions.
- Implemented the perspective transform using the matrix stacks. Now 3D objects that are further away look smaller on the screen. I used the question and answer site Stack Overflow to help me understand and implement the transform.
- Tried drawing face indices onto the grid in order to help me debug the code.

Sources used:

- "Manipulating the Matrix stacks" OpenGL programming guide (red book) http://www.glprogramming.com/red/chapter03.html#name6
  - I wanted to find out what matrix stacks were.
  - This source gave me a good explanation of how to use matrix stacks. However, the code examples were confusing and didn't match up with anything I had seen in OpenGL tutorials before. After some research it turned out that the code examples were outdated and used deprecated functionalities (immediate mode) from an old version of OpenGL. I am going to ignore the code examples and just focus on the explanations.
- "glm::perspective explanation" Stack Overflow http://stackoverflow.com/questions/8115352/glmperspective-explanation
  - This answer explained what the perspective function in glm does, and what the matrix constructed means. I wanted to use the function in order to create perspective in my 3D scenes, but I was unsure about the parameter meanings and the official documentation did not help. This answered my questions.

17/08/13 *(3 hours)*

- Started implementing creases into the mesh engine. This allows values (such as texture UV) to change discontinuously. I am using the approach described in this blog post. It is not completely working.

23/08/13 *(3 hours)*

- Started implementing a bridge simulation demonstration, inspired by "Bridge Builder". This is using my existing Verlet integration physics engine, which I have already shown to work. Created node/link system, to represent a bridge, where nodes are the joins between two or more segments of material, and links are the segments of material themselves. This is similar to a graph in mathematics.

24/08/13 *(2 hours)*

- Continued working on bridge simulation.
  Worked on the user interface side of the demonstration, making mouse and keyboard work properly.

28/08/13 *(1 hour)*

- Continued bridge simulation demonstration.
  In addition to bridge nodes, I needed to add simulation nodes too, as a link that is drawn by the user cannot be simulated as one single rigid body. It must be split up into smaller components each of which are simulated individually. These are joined together with simulation nodes.

4/09/13 *(3 hours)*

- Continued working on bridge simulation.
- I implemented the length constraint in the physics engine.
- I implemented the "cable" material, which is flexible and has high tensile strength.

13/09/13 *(6 hours)*

- Continued bridge simulation demonstration.
- Found out how to calculate angle between two vectors from Stack Overflow.
- From this I worked out how to implement an angle constraint, to create inflexible materials such as "iron". *To do*: iron out the bugs and get some feedback.

Sources used:

- Signed angle between two vectors without a reference plane Stack Overflow
  http://stackoverflow.com/questions/10133957/signed-angle-between-two-vectors-without-a-reference-plane/10145056#10145056
  - This source gave me the formula for working out the angle between two vectors. I could not find this in my FP4 textbook. The "accepted" answer on the site has some limitations which this answer avoids, by using atan2 rather than acos.

14/09/13 *(5 hours)*

- Fixed glitches in the angle constraint.
- Finished and polished bridge simulation demonstration.
- Published online, and linked on my Facebook. I got a lot of feedback from my friends.

15/09/13 *(2 hour)*

- Tweaked bridge simulation based on feedback from friends.
- Writing source evaluations.

18/09/13 *(3 hours)*

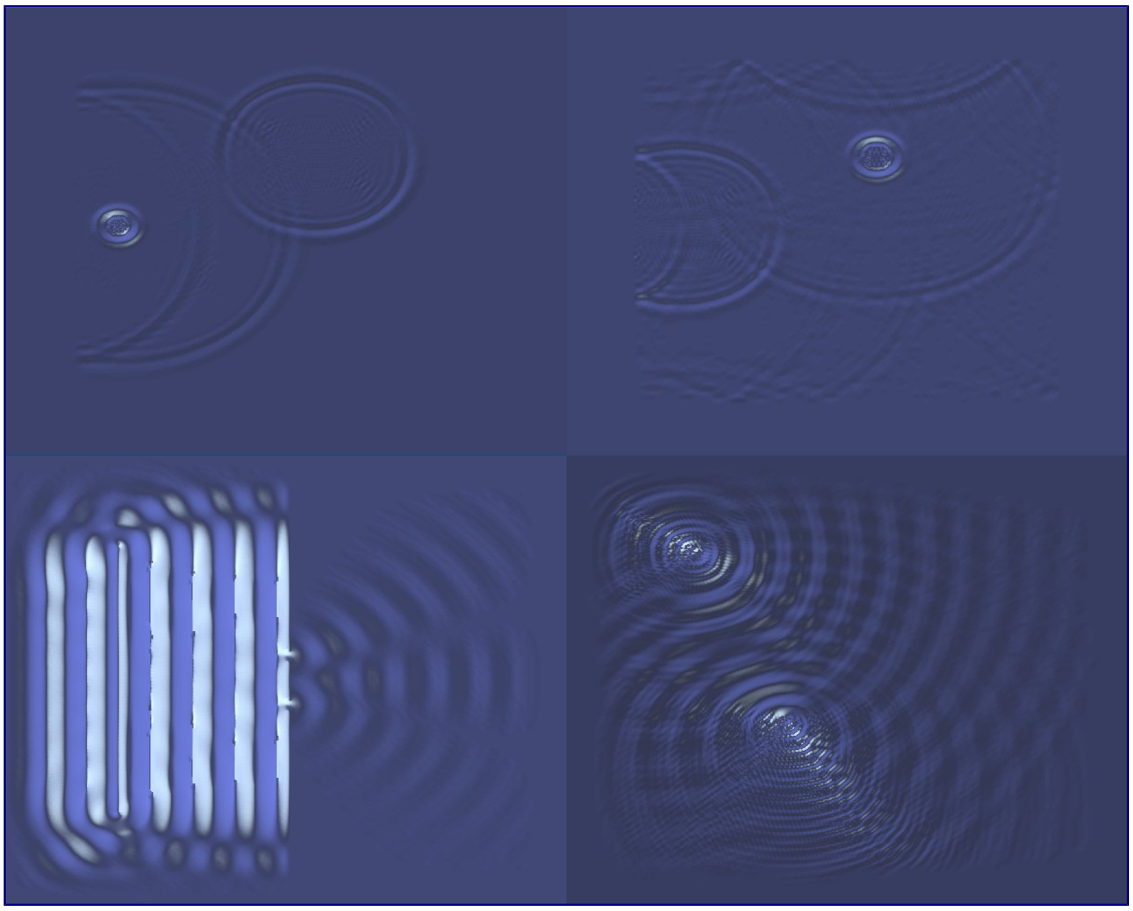
- Started implementing diffuse lighting, based on the cube experiment that I had written earlier.
- Researched Blinn-Phong lighting.

Sources used:

- Blinn-Phong model Jason McKesson http://www.arcsynthesis.org/gltut/Illumination/Tut11%20BlinnPhong%20Model.html
    - This source gives me how to implement the Blinn-Phong model using shader code. It corresponds to the diffuse shading I used for my cube demonstration earlier, but adds specular shading as well, adding to the realism.

27/09/13 *(4 hours)*

- Completed implementing lighting on my wave demonstration. It uses Blinn-Phong shading based on the Phong reflection model.



- Assembling the bibliography for my project.

06/10/13 *(5 hours)*

- Writing source evaluations and connections.