# UFO and COws

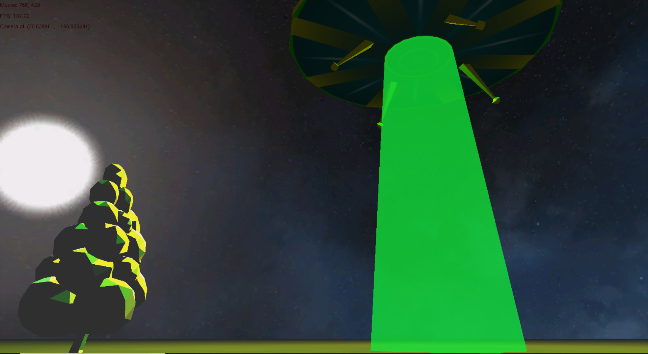
# CMP 203 Graphics programming - Jake Bretherton

## Controls

The scene has two main camera modes, following the spaceship and a free flying camera.

|  |  |  |
| --- | --- | --- |
|  | Operation | Key |
| Camera mode: | Free camera | V |
|  | Follow spaceship | C |
|  |  |  |
| Camera controls: | Left/Right | A/D |
|  | Forward/backwards | W/S |
|  |  |  |
| Free camera only: | Up/Down | E/Q |
|  | Look | Mouse |
|  |  |  |
| Spaceship only: | Abduct Cow | Spacebar |
|  |  |  |
| Other/Debug: | Wireframe | R |
|  | Unlock/lock mouse | M |
|  |  |  |
|  |  |  |

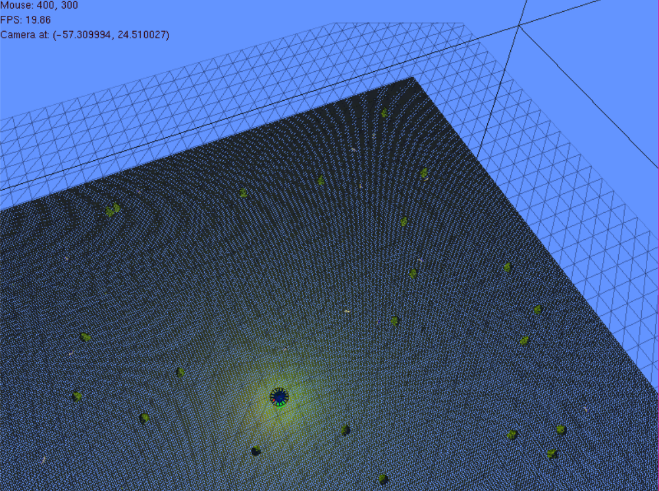
## Oveveiw



In this scene you pilot a UFO that is very interested in abducting cows. The scene consists of a variety of self-made and public domain models and textures, multiple lighting techniques and hierarchical model structuring. It also contains depth-sorted alpha blended transparent objects, user interaction as well as a handful of more advanced techniques.

## Geometry

### Procedural geometry

I generated two planes, one at a high resolution with a grass texture and one at a lower resolution with transparency - to be water surrounding the grassy plane.

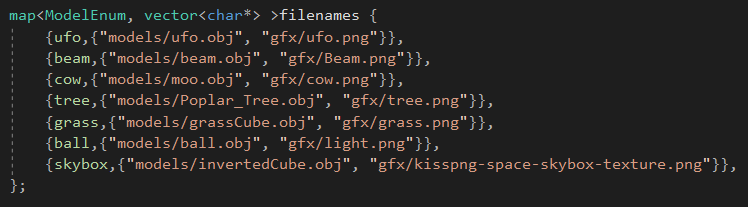




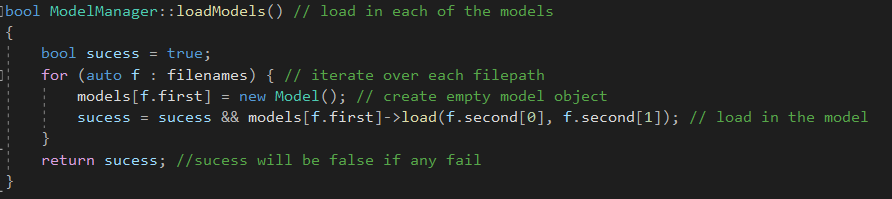
Procedure for generating plane given x and y size and the resolution of the plane.

### Model Loading

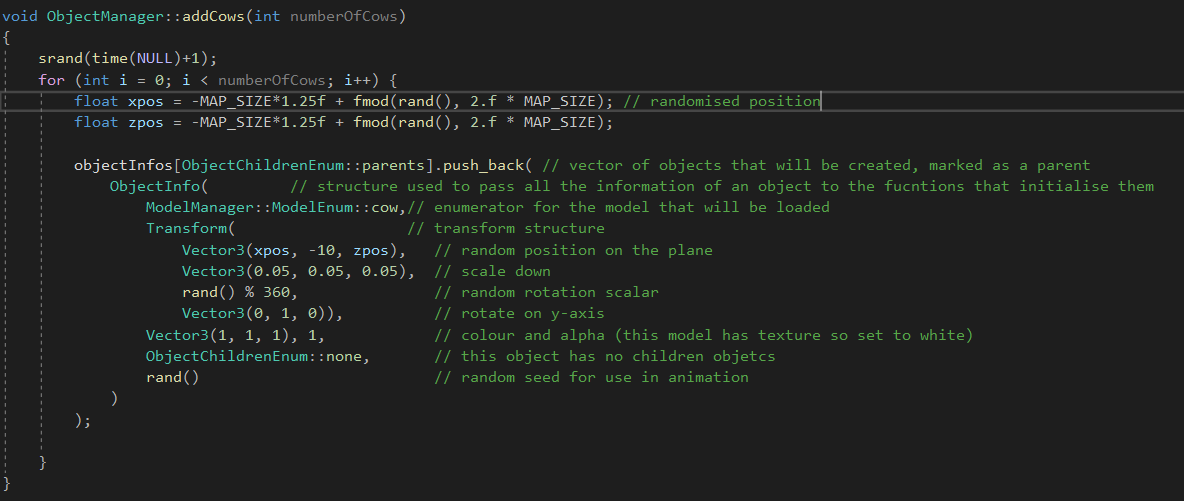
Initially one of each of the objects that appear in my scene are loaded in by the model manager

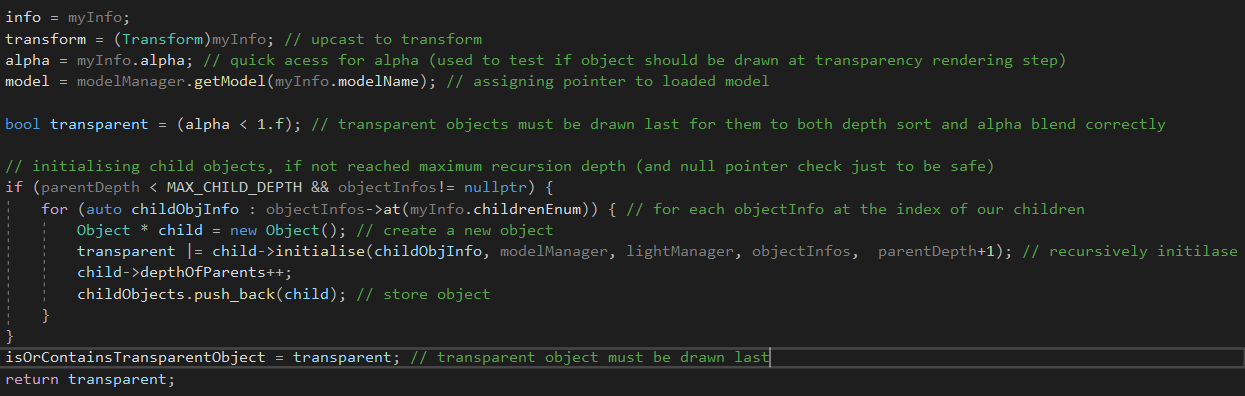


List of the file paths for the objects

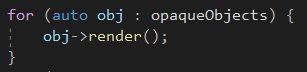


Function that initialises models

I use this function to add a load of ‘Objectinfos’ to the map that will initialise all of the objects

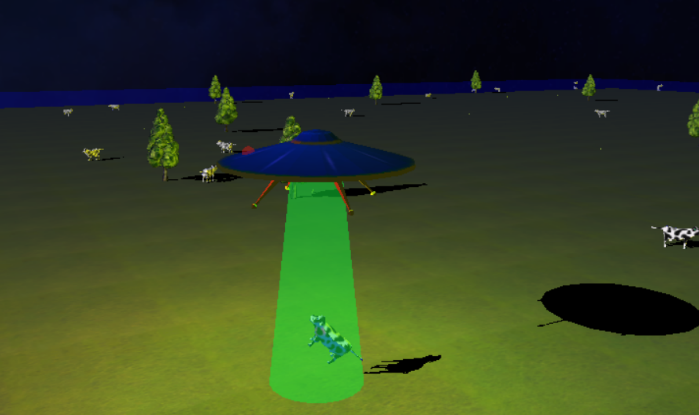
This function pushes back to a map vector of all the objects that will be initialised, when this function is called for each object

The model for each object is simply indexed from the model manager and a pointer to the model object is stored in the object. This pointer is then dereferenced at the time of rendering the model.



## Camera

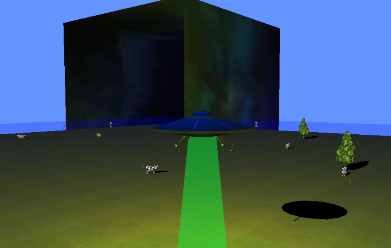
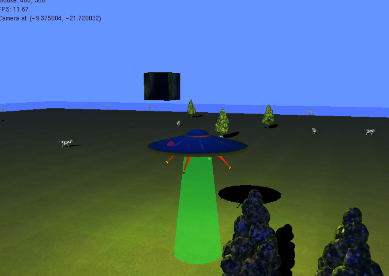
### Ufo Control

The camera follows the UFO and is controlled though wasd and mouselook. As the player moves through wasd the camera and ufo remains on a level plane, the ufo is moved up and down with the mouse. The camera only has 50 degrees of vertical freedom in this mode to prevent the spaceship from clipping through the ground. The camera is bounded to the scene and cannot fly the ship past the edge of the plane. This camera mode can be entered by pressing the ‘C’ key

### Free cam

The camera can also be controlled separate from the ufo for less bounded control, this mode is entered by pressing the ‘V’ key. In this mode the camera will move forward in the direction looked in with W/S, regular left and right with A/D and can move up and down with Q/E. The camera is not bounded by the edge of the scene and vertical look is bounded to 160 degrees.

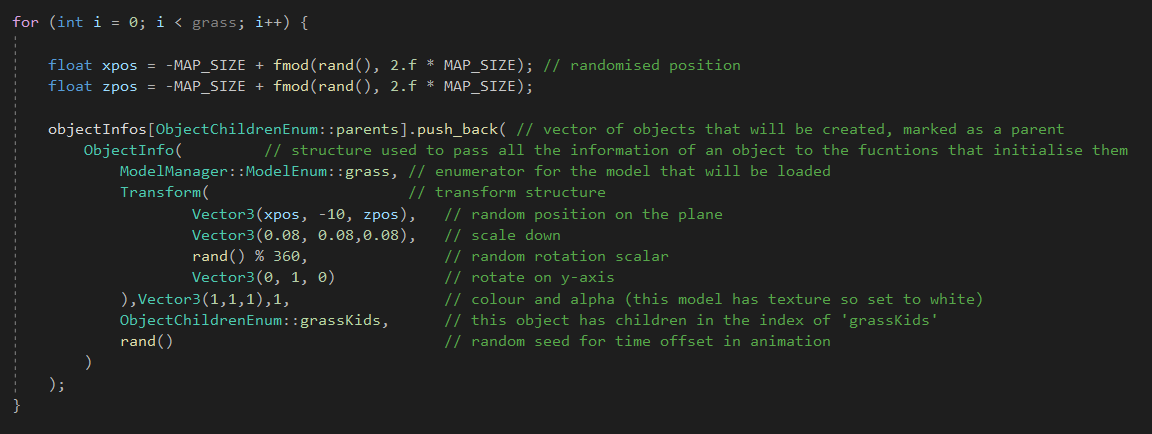
### Skybox

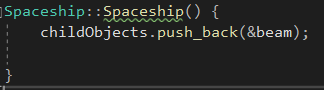


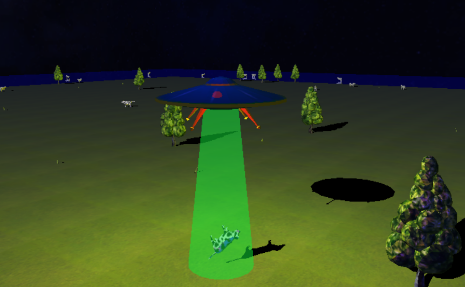
The skybox is drawn to be a small cube around the camera with its face’s normals inverted. This is drawn first and is not drawn to the depth buffer, so all rendered objects will be drawn over it, effectively faking each wall being infinitely far away – as far as depth sorting goes. This cube is moved around with the camera. This allows for a detailed sky that can never be walked past.

## Hirearchical Object Structure

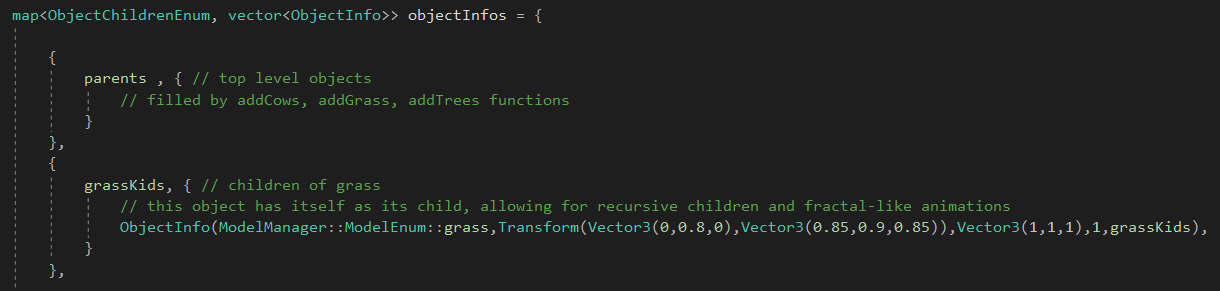
### Object Structure

The scene’s elements are made up of ‘Object’ object class types. Each object can also contain a number of children objects.

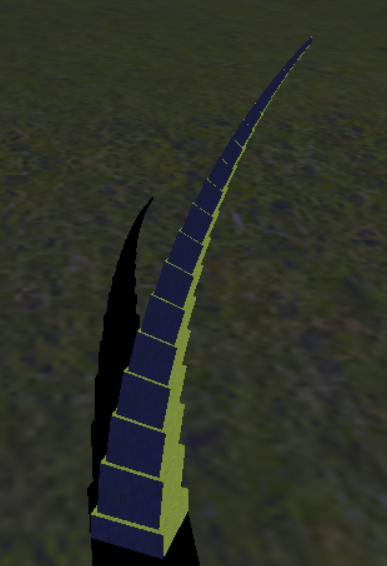
For example the spaceship has its beam as a child object

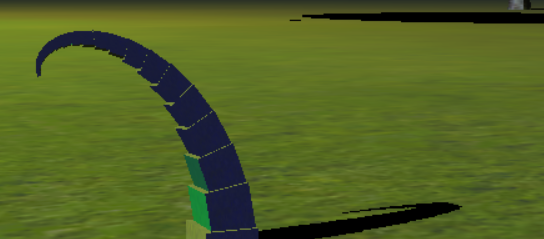


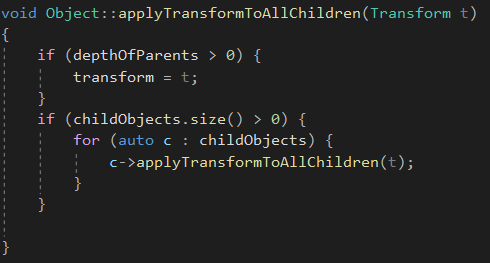
This allows the beam to stay attached to the spaceship without having to update it’s position to be the same as the UFO as transforms of parents objects are aplied before child objects’

This object here, a blade of grass, is its own parent and child. It has a recursive relationship with its parenting object allowing easy animation of it bending as each chid inherits its parent’s render transform

### Recursive rendering

The parents transform is applied and is only popped after the child’s transform has been applied, meaning child object’s position, scale, rotation etc. are in the local space of the parent; you only need to alter the parent for each child to follow.

Here the grass is actually made up of simple cube meshes, with each parented to the one below it. This allows incredibly smooth animation with relatively simple code.



This recursive property allows us to apply a small transform to all child objects an object contains and have that transform cascade down in all the children

## Lighting

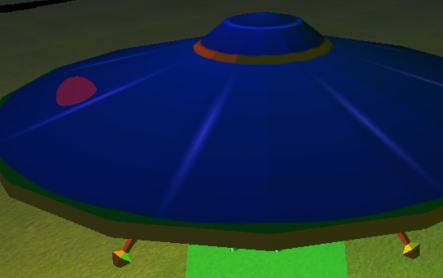
The lighting is applied after the scene has been rendered. The light contains information about its position, intensity, colour, direction, ambient information and its type. The scene has three major lights in it:

One directional:

This light has no position; only direction. In this scene it also handles all of the ambient light.

One point

This light has only a position; no direction. This light is animated.

A halo is drawn round it so players can see where the light is coming from.

One spot

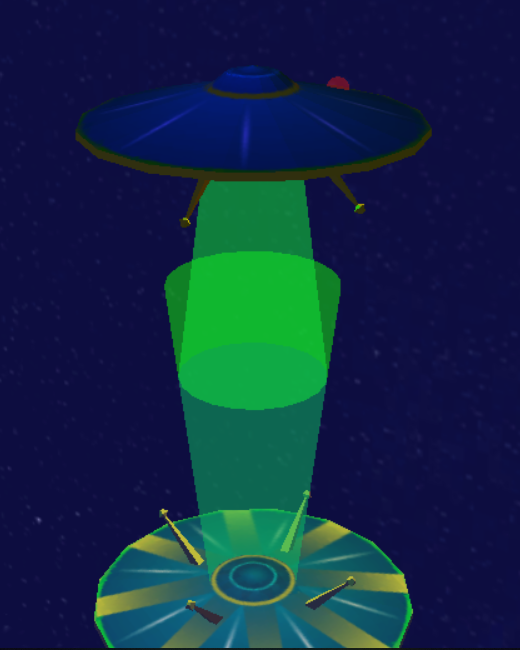
This light has position and direction, as well as a radius of its frustrum.



The light manager class handles lighting (as there are only eight possible lights a scene can have, a manager class keeps track of which light object belongs to which light).

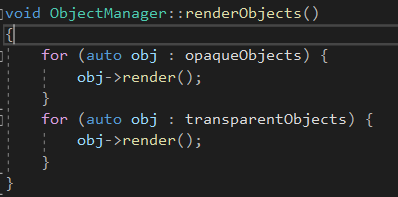
I use this function to apply each light to the scene, using a light class that stores information about each light such as its position, colour, intensity etc.

## Transparency

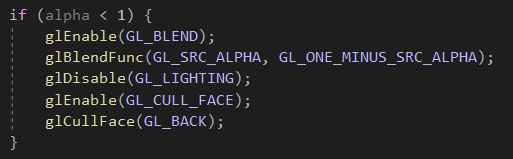


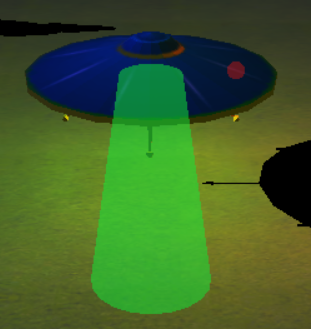
Transparency blending must be carefully managed.

The scene contains a handful of transparent objects, such as the ufo’s beam, red blinking light and the water surrounding the grassy planes.



For transparency blending to work properly two things must be done, alpha blending must be enabled, and the transparent objects must be drawn last, from back to front. This is because any object that has transparency needs to blend with all objects behind it, but objects behind will not be drawn if the transparent object is rendered first because they will fail the depth test against the transparent object.





You cannot simply disable the transparent objects from writing to the depth buffer, as this prevents other objects from correctly obscuring them

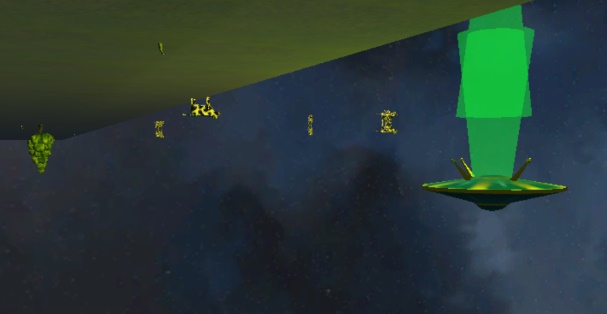


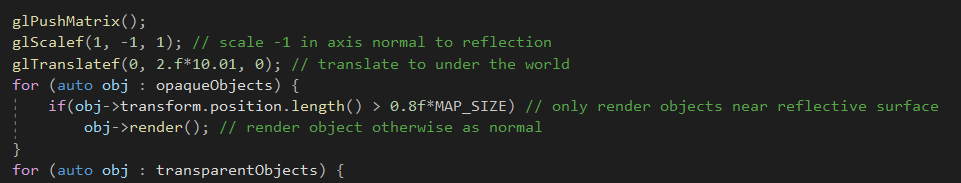
You also must cull back faces of 3D transparent objects to prevent this rather ugly seam effect.

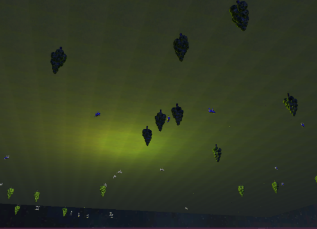
## Reflection

Reflections, like most things in graphics programming, is a technique where the effect is completely faked.

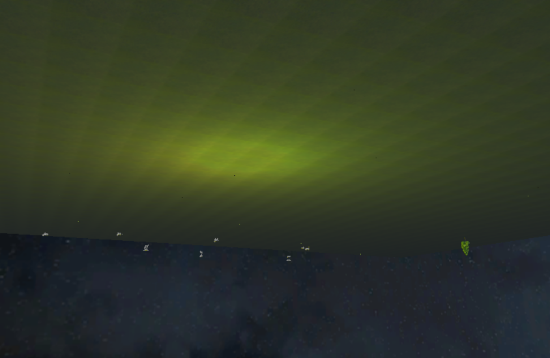
The scene around reflective objects is rendered again at a -1 scale in the axis normal to the reflective surface.

We also have to use the stencil buffer to make sure objects are correctly being displayed - more on that later.



We only render the objects close to the reflective surface as objects far away will never be visible and so would be a waste to draw.

Drawing full scene mirrored

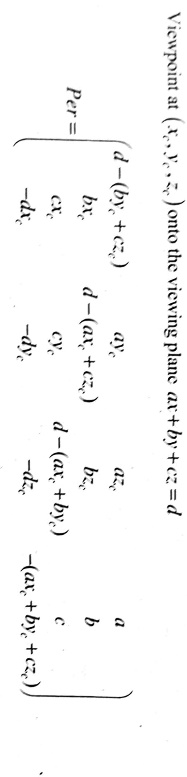


Only drawing the objects close to the water

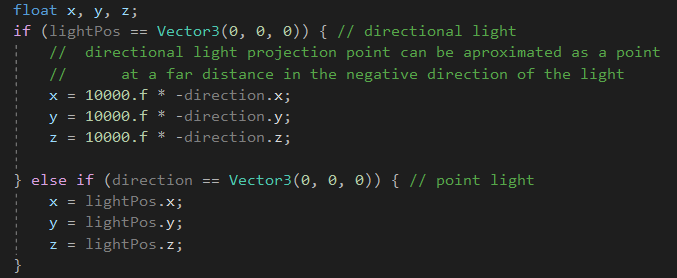
## Shadows

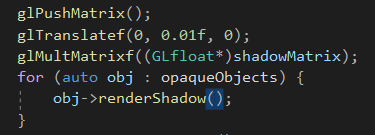
There are a couple of different techniques for creating shadows in games, I opted for planar projected shadows as it provides very good-looking shadows and is not especially complex when projecting onto a flat plane like in my scene. This method transforms a model onto a flat plane creating a shadow.

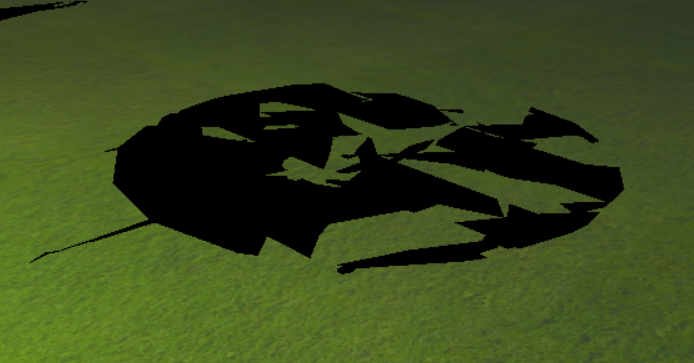
In the interest of performance, I opted to only project one shadow per object - from the main directional light in my scene.



This formula is used to generate the projection matrix. This however works with a viewpoint (being the location of the light) which directional lights do not have, so I decided to fake a point light at far distance rather than write separate functions.





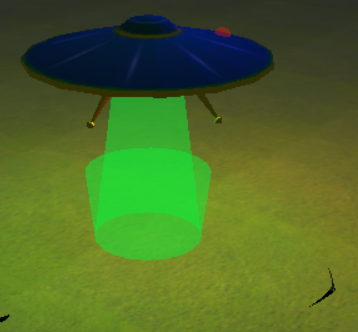
Before applying the shadow matrix, we move the models up a touch so that the shadows don’t z-fight with the ground plane

## Stencil Buffering

The stencil buffer allows us to draw models to a separate buffer than the one that will be displayed, and then use this to only draw things that pass its test. For example, render a plane to the stencil buffer then use this to only render elements of a scene that are inside that plane, or outside it.

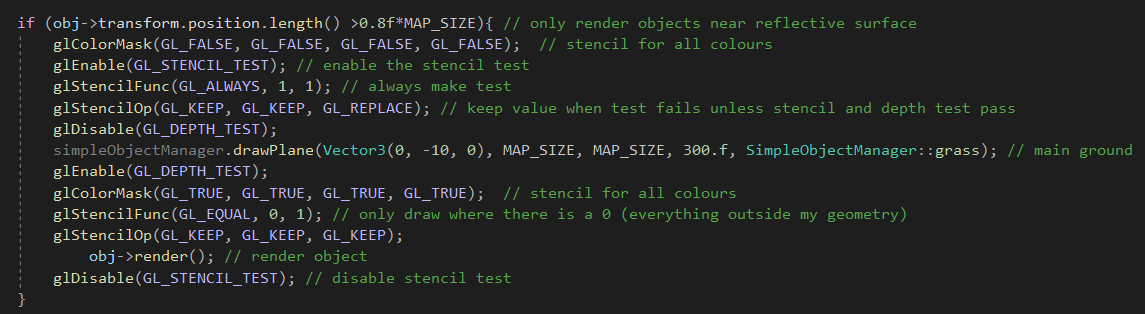
### Reflection culling

In my scene, the beam from the reflected spaceship intentionally passes though the floor at some points over the water, as the beam represents light it would naturally reflect in this way, coming back up out of the water.



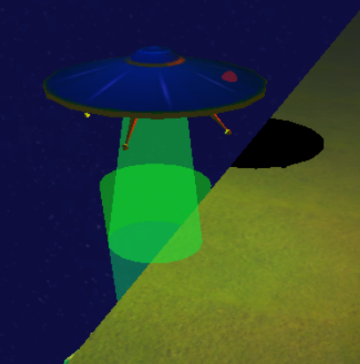
However, we definitely don’t want this to happen over land, so I made use of the stencil buffer so that the beam is only drawn over the water.

For this test I render the grass plane to the stencil buffer so that it writes 1s where the geometry is, and then set the test to pass whenever there is a 0, or where the geometry is not





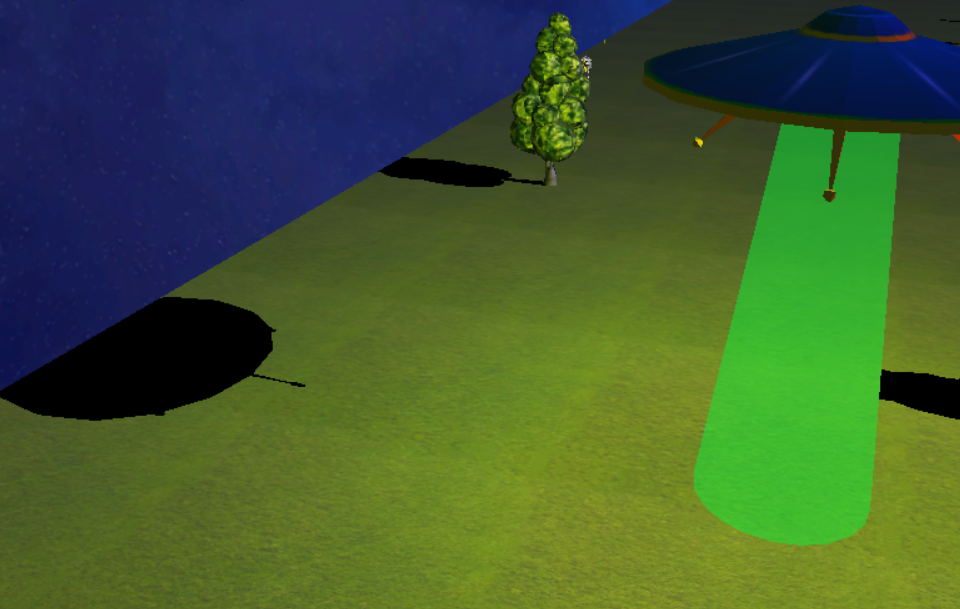
This is without the stencil test



This is with the stencil test and culling

### Shadow culling

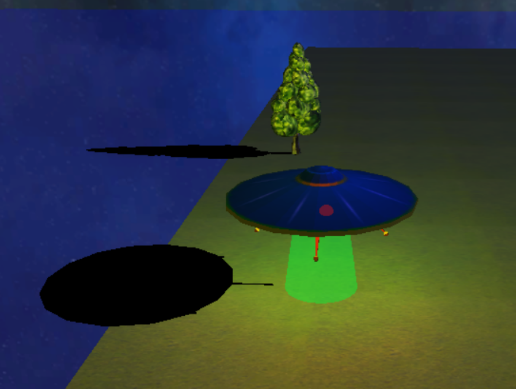
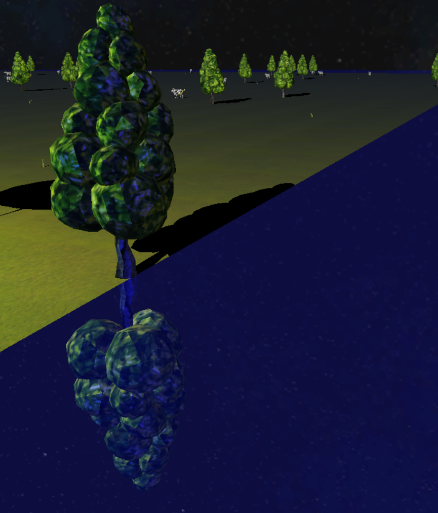
Conversely to the reflection culling, we want shadows to cast only onto the ground and not onto the water.



For this we generate the same stencil mask but instead of passing the test on a 0, we pass the test on a 1

glStencilFunc(GL\_EQUAL, 1, 1); // draw only where there is a 1 (where the geometry is)

This is how it looks without applying this effect, compared to with this effect



## Bibliography

Models:

|  |  |
| --- | --- |
| Tree | <https://www.turbosquid.com/FullPreview/Index.cfm/ID/1189852> |
| UFO | <https://www.turbosquid.com/FullPreview/Index.cfm/ID/1333537> |
| Cow | <https://www.turbosquid.com/FullPreview/Index.cfm/ID/1027529> |

Textures:

|  |  |
| --- | --- |
| skybox | <https://www.cleanpng.com/png-space-skybox-texture-mapping-cube-mapping-night-sk-776480/> |
| Cow texture | <https://www.vectorstock.com/royalty-free-vector/black-and-white-cow-skin-animal-print-seamless-vector-9959490> |
| UFO texture | <https://www.turbosquid.com/FullPreview/Index.cfm/ID/1333537> |
| Tree leaves | <https://depositphotos.com/264439756/stock-photo-green-detailed-summer-tree-leaves.html> |
| Tree bark | <https://isorepublic.com/photo/tree-bark-texture/> |
| Grass | Texture provided by Abertay |

All other textures and models I created myself.