# Real-Time Shadows Report

# Description of shadow algorithms

## Planar shadows

Planar shadows are a very simple implementation of shadows. According to Kainulainen (2004), in planar shadows, the vertices of an object are projected onto a flat plane from the direction of the light. Both the light vector and the normal to the plane must be specified in order to create the shadow projection matrix. Once this has been created, the object must be drawn twice; once as normal, and once transformed by the shadow projection matrix onto the flat plane. The rendered shadow can be set to render with a darker colour.

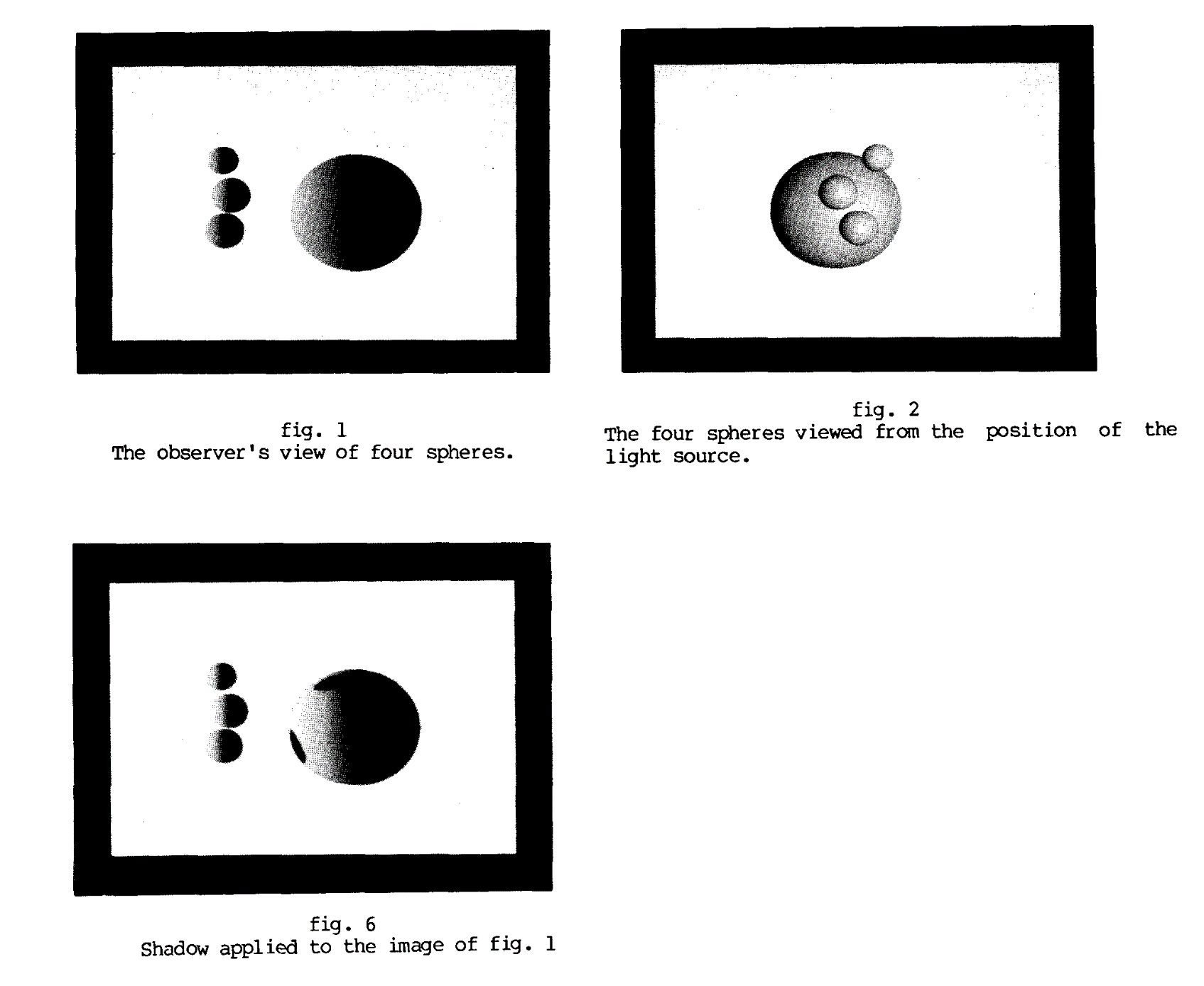
Planar shadows are a very simple implementation with many limitations, e.g. inability to cope with curved surfaces, the need to create separate shadows for every object in the scene etc. They are rarely used except for where only very simple shadows are required.

## Shadow mapping

Two of the more common algorithms used to implement dynamic shadows are shadow mapping (which was used in this implementation), and shadow volumes. Shadow mapping is used to add shadows to an existing 3D scene. The algorithm was first presented by Williams (1978). The main goal of the algorithm was to enable shadows to be cast on curved planes, as opposed to the flat planes required by simpler methods like planar shadows. The first step in shadow mapping is to construct a view of the scene from the perspective of the light source. Williams states that it is not necessary to calculate the shading values (colours); only the Z-values for each object are required. This data will be stored in the Z-buffer (or depth buffer).

The next step of the algorithm is to construct a view of the scene from the perspective of the “camera”, or observer viewpoint. Whenever the values for a pixel in the scene are calculated, it is necessary to transform the pixel’s X, Y, Z coordinates into light space and compare against the depth buffer, in order to check if the pixel is visible from the light source. If the pixel is not visible to the light source, then it is in shadow and should be shaded darker than normal. If the pixel is visible to the light source, then it is shaded as normal.

Williams (1978) provides an image of the output of the shadow mapping algorithm. The first image shows the observer’s view of the scene, the second image shows the image rendered to the depth buffer from the perspective of the light, and the third image shows the final output, with the shadowed regions calculated based on the pixel visibility.



## Shadow volumes

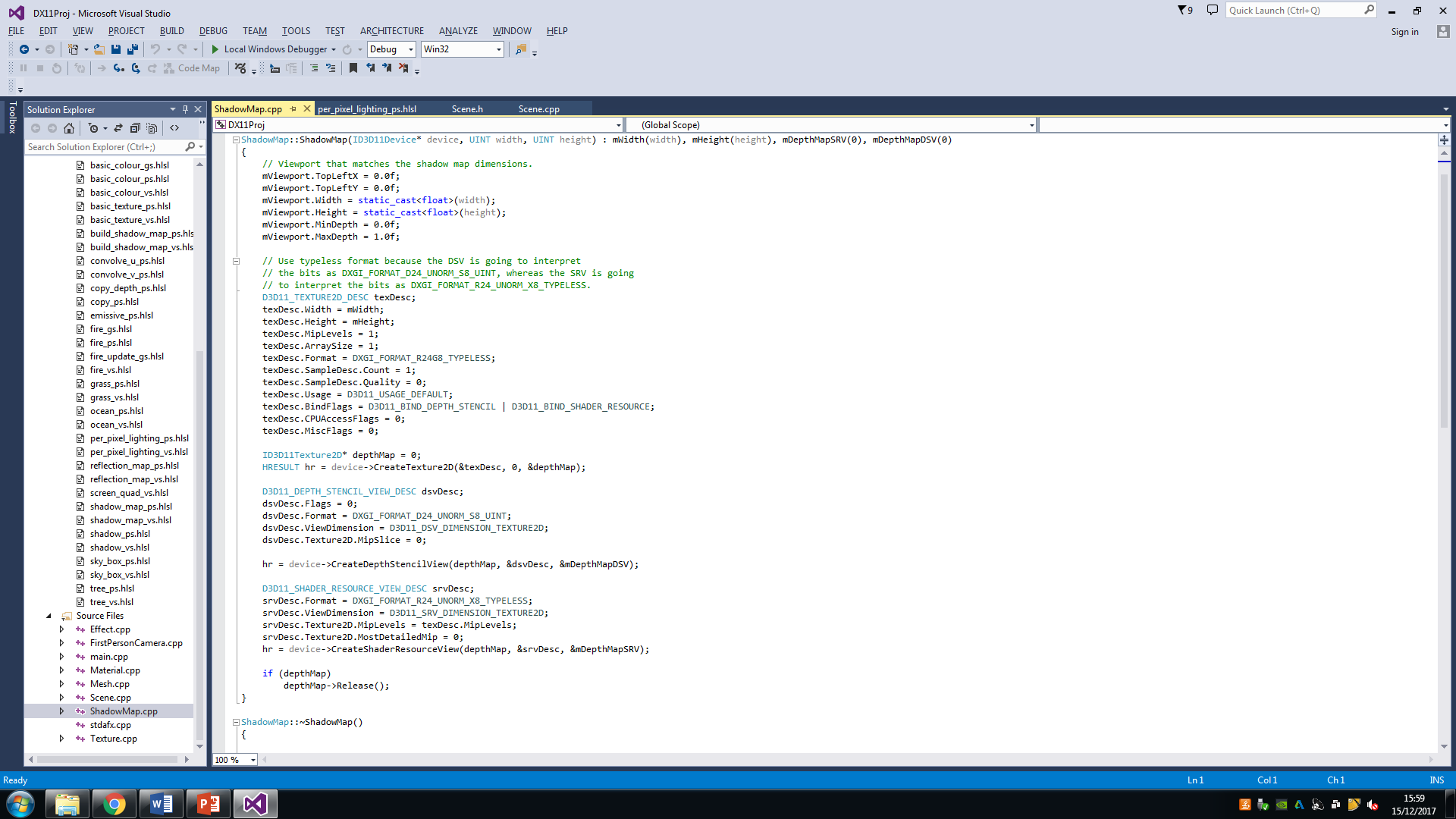
Shadow volumes were first introduced by Crow (1977). According to Tsiombikas (no date), the algorithm consists of two main parts: building the shadow volumes, and rendering shadows. A shadow volume is a polygon mesh that stores the boundaries of the shadow cast by an object. A shadow volume must be built for every object in the scene. If there are multiple lights in the scene that cast shadows, then shadow volumes must be built for every combination of light and object. In order to build a shadow volume for an object, the contour edges of the object must be found, as seen from the light’s perspective. A contour edge is defined as an edge that has one adjacent edge facing towards the light, and another facing away from it.

Once the algorithm has found every contour edge of an object, it will extrude them away from the light by a certain distance (defined by the programmer) in order to build the shadow volume. At this stage, the shadow volume will be incomplete, because it will not take into account the boundary where the shadow is touching the object casting it. This is resolved by adding the relevant polygons from the object to the shadow volume.

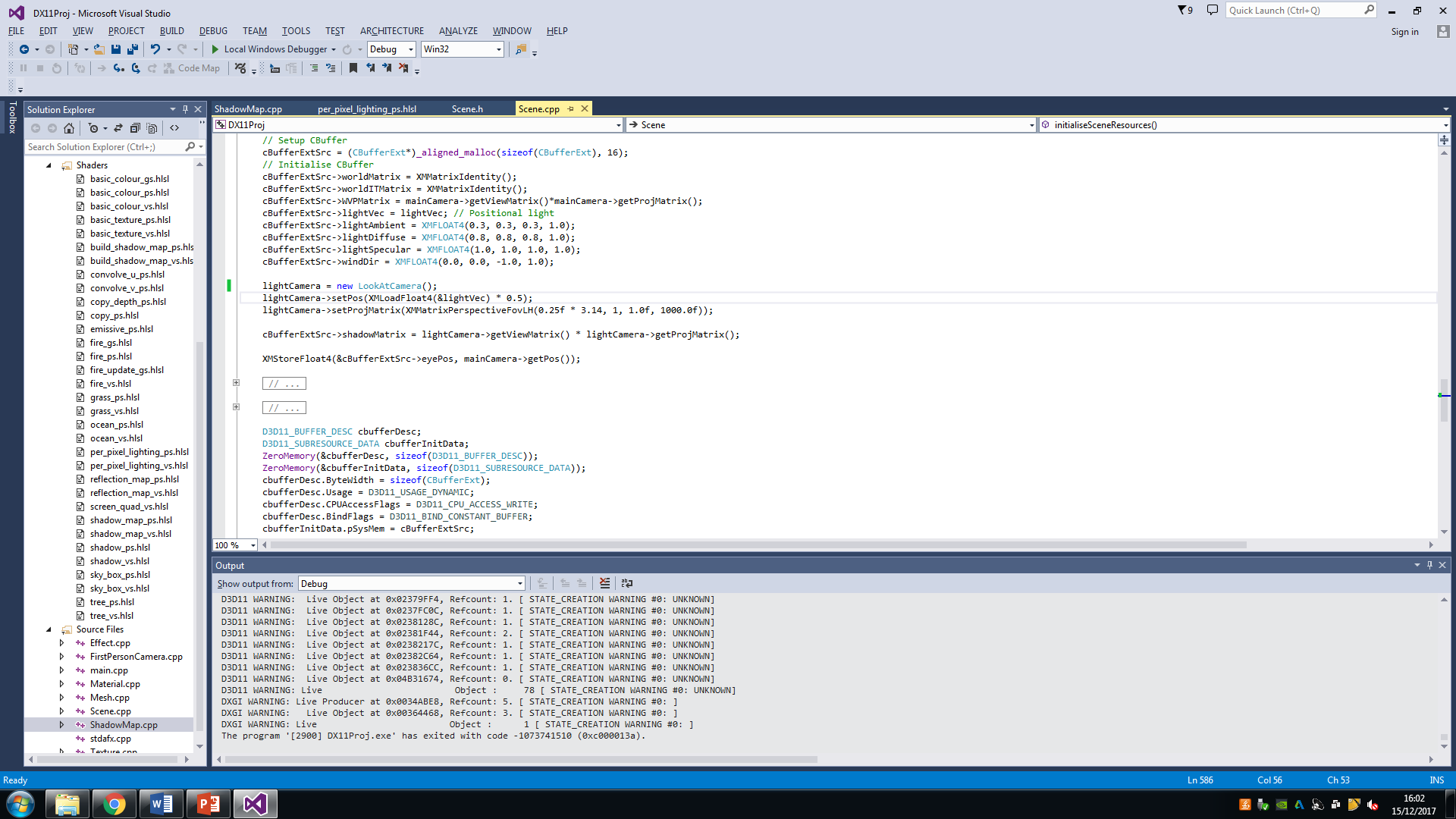
The next step is to actually render the shadows. In order to determine whether a pixel is in shadow or not, a ray is cast from that pixel through the scene. Every time the ray passes through the front face of a shadow volume, a counter is incremented. Every time the ray passes through the back face of a shadow volume, the counter is decremented. If the counter is greater than zero at the point where the ray comes to a stop, then that ray has not passed through the back face of one of the shadow volumes, and therefore the pixel lies in shadow. If the counter is equal to zero, then the pixel is not in shadow, and can be rendered normally.

# Description of shadow algorithm implemented

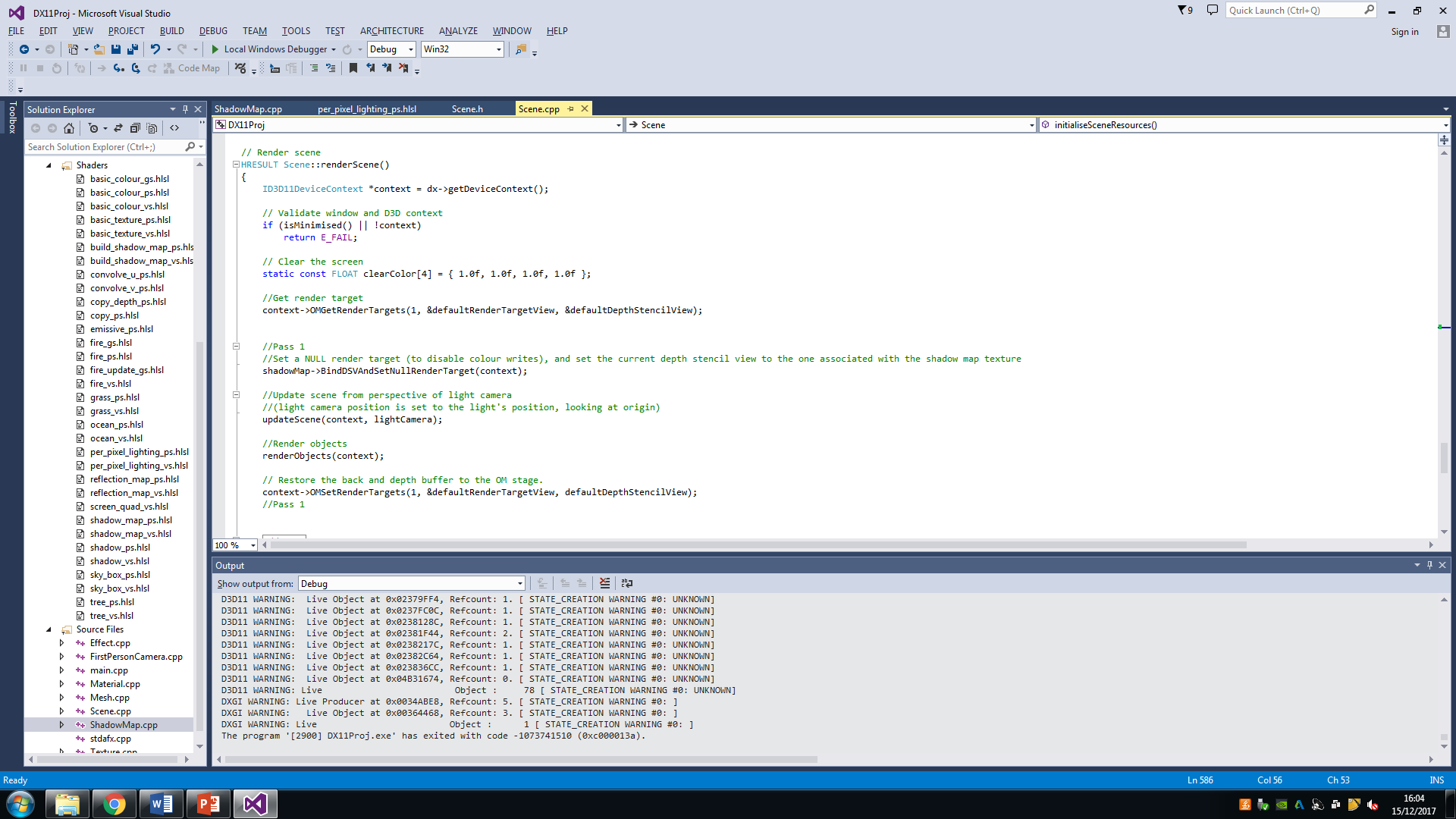
The shadow mapping algorithm implemented in this scene is based on the version presented by Frank Luna (2012). First, it is necessary to set up the shadow map itself. This involves setup of a viewport, 2D texture description, depth stencil view description, and shader resource view description. A depth stencil view and shader resource view are then created for the shadow map texture. See below.



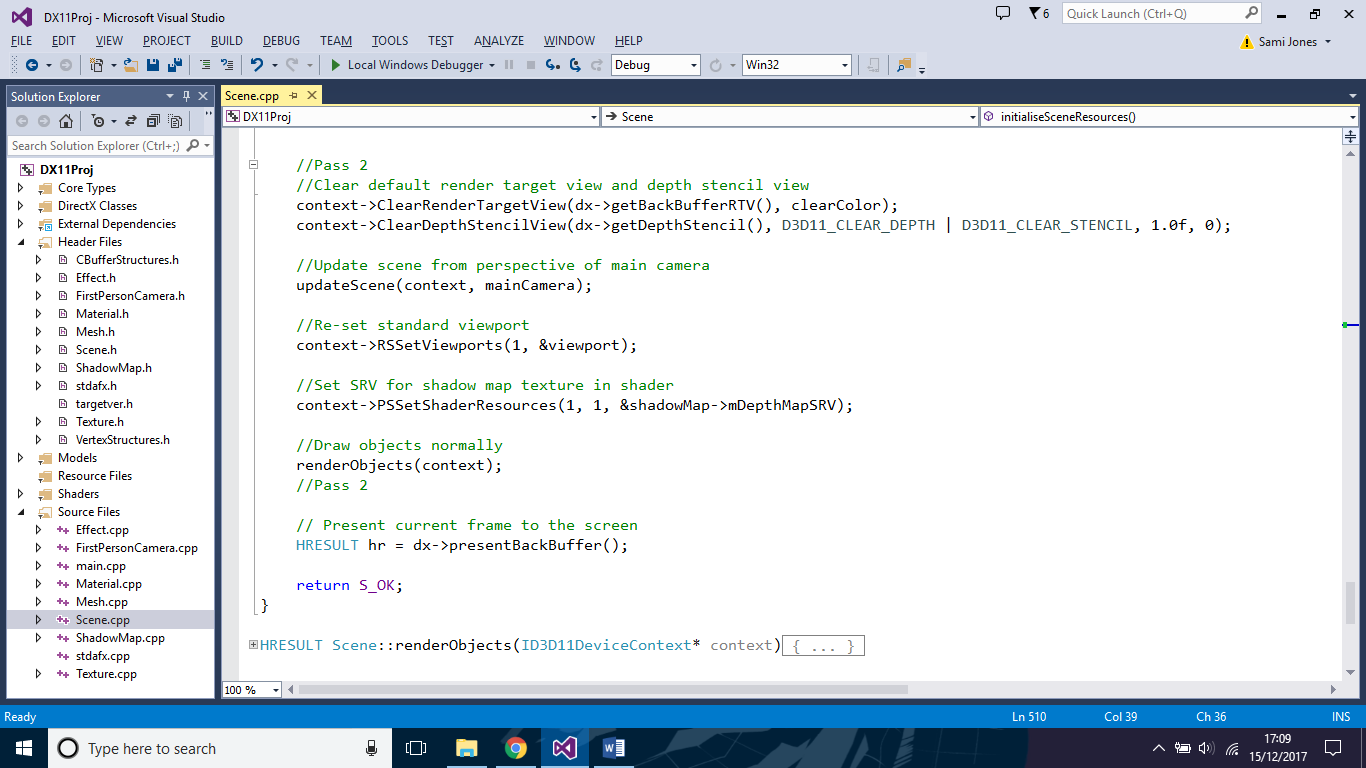
Next, it is necessary to set up a camera positioned at the light, looking towards the scene. See below.



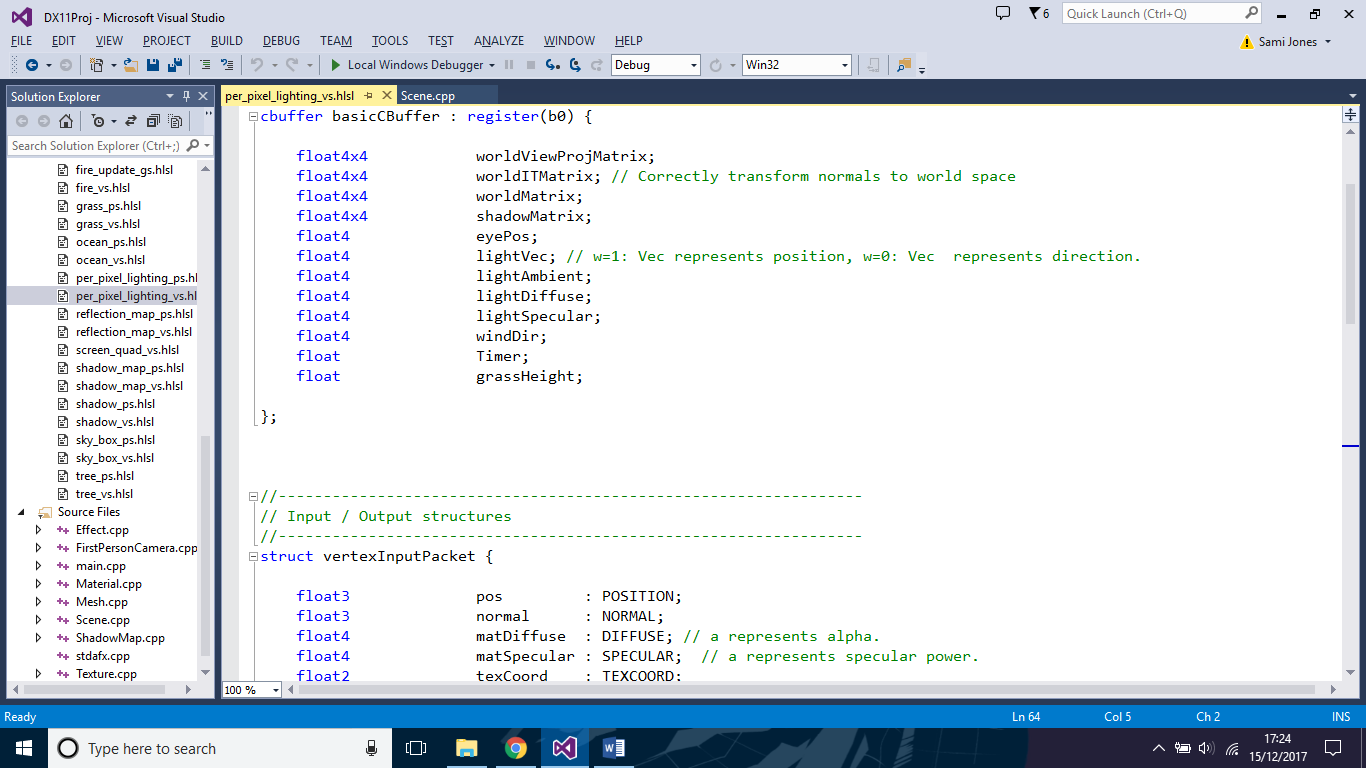
The scene is drawn with shadows using two render passes. In the first render pass, a null render target is set to disable colour writes, and the current depth stencil view is set to the one associated with the shadow map texture. Only the depth stencil view needs to be set, because we only want to render the depth to the shadow map texture, not the colours of objects. The next step is to render the scene from the perspective of the camera positioned at the light. Once this is done, the render target can be reset to the default render target view and depth stencil view (the ones used for displaying to the screen). See below.



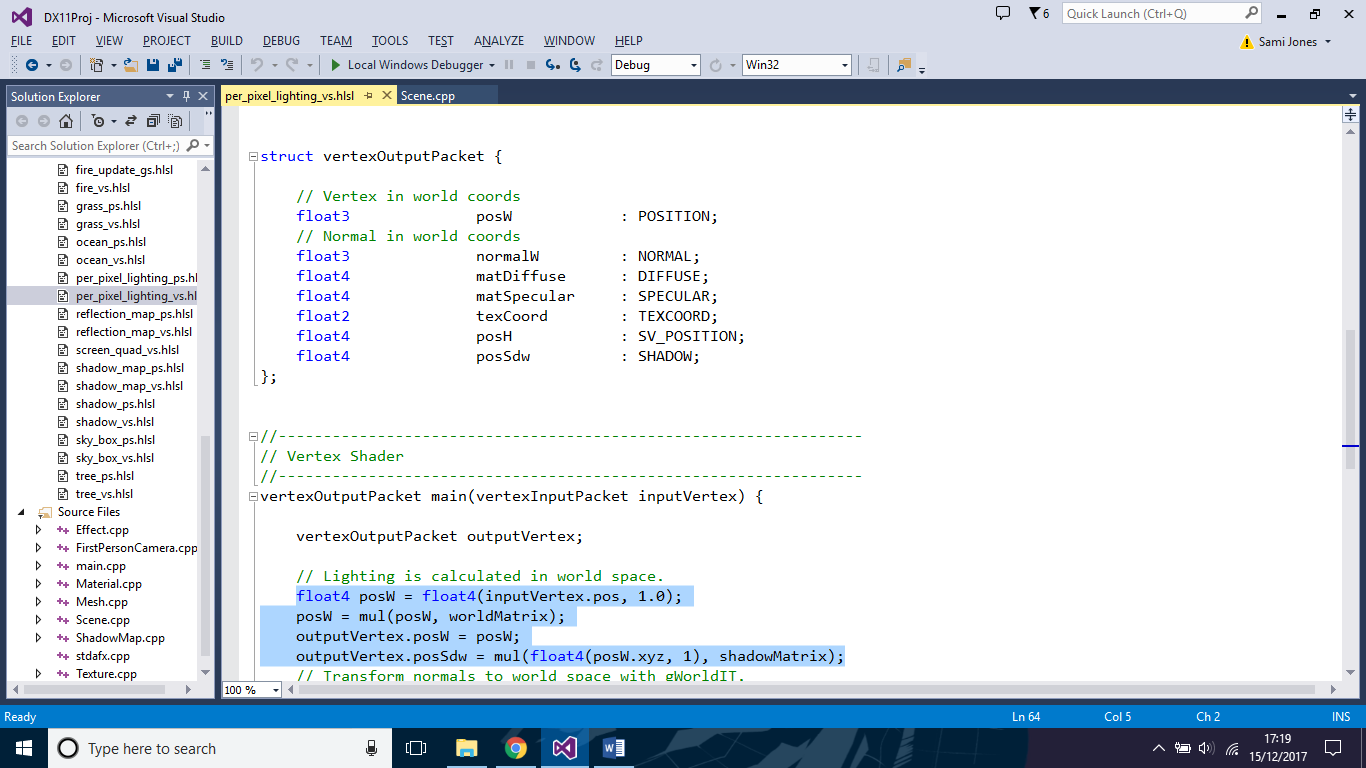
In the second render pass, the render target view and depth stencil view for the screen are cleared so the scene can be drawn again. The standard viewport is set, as the screen differs in size from the shadow map texture. Also, the shadow map created in the first render pass is set as a shader resource (in addition to the standard texture resource that already exists in the shader). The scene is then rendered a second time, but this time from the perspective of the main camera (or observer). See below.



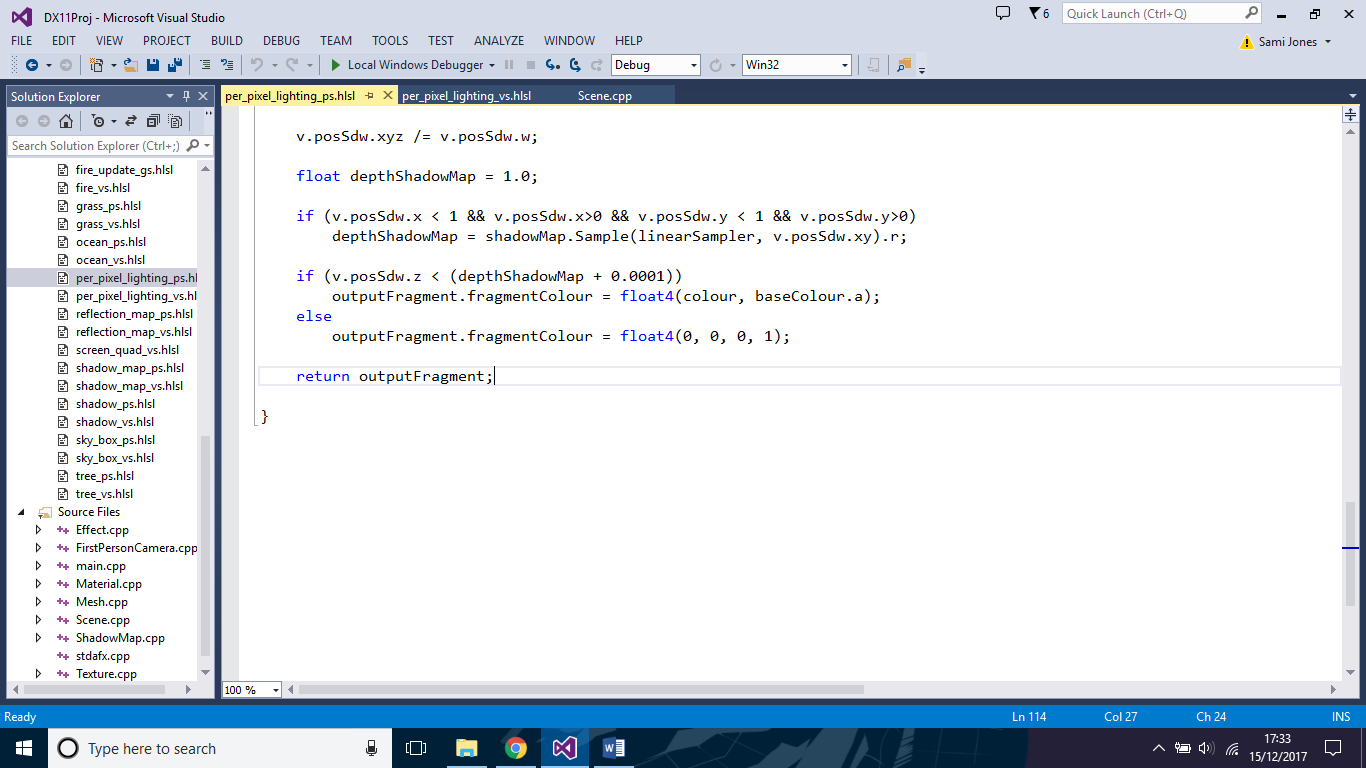
When rendering the scene in the second render pass, the per-pixel lighting shaders perform some extra calculations to check whether each pixel is in shadow or not. In order to do this, the vertex shader takes in the shadow transformation matrix (created when the scene is updated) as part of the data in the constant buffer. See below.



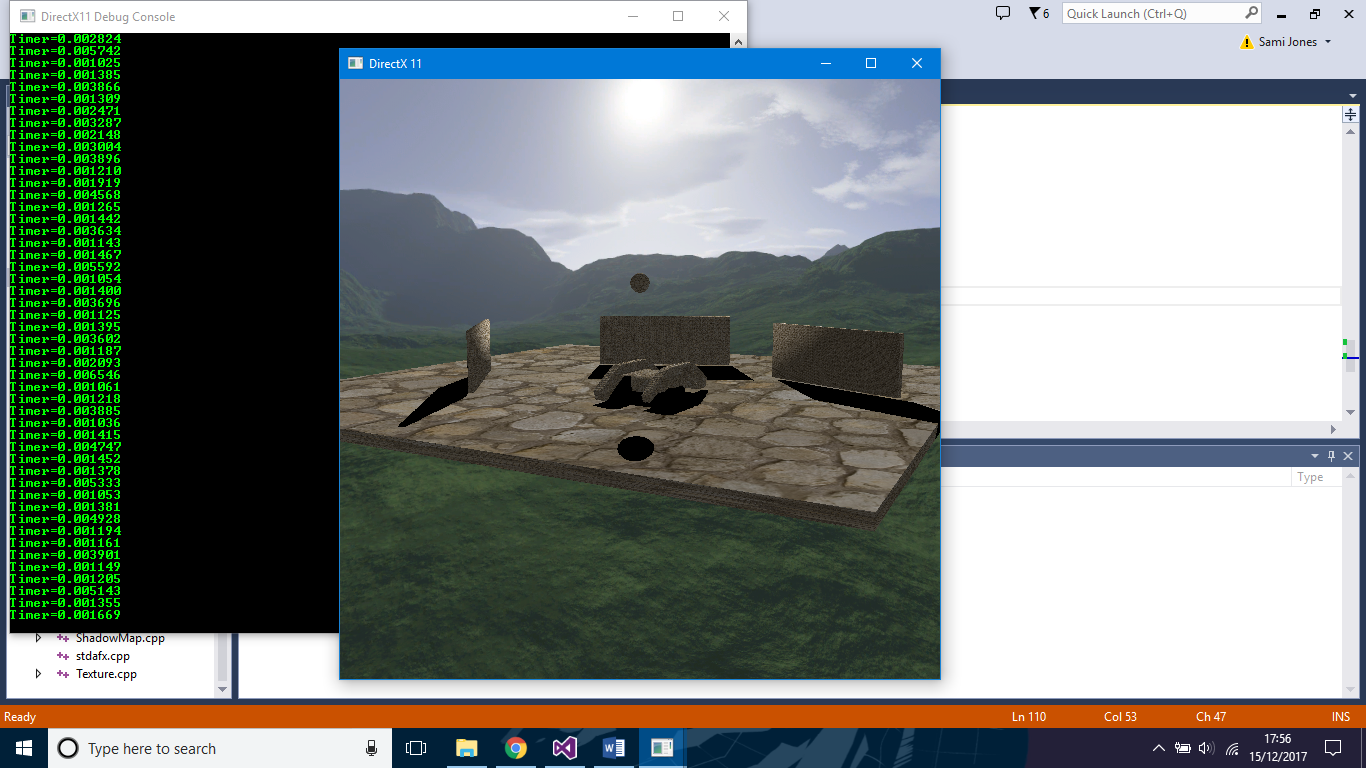
The vertex shader uses the shadow matrix to transform vertex coordinates from the perspective of the main camera to the perspective of the light camera. The transformed coordinates are stored in the posSdw variable and passed to the pixel shader as part of the vertex output packet. See below.

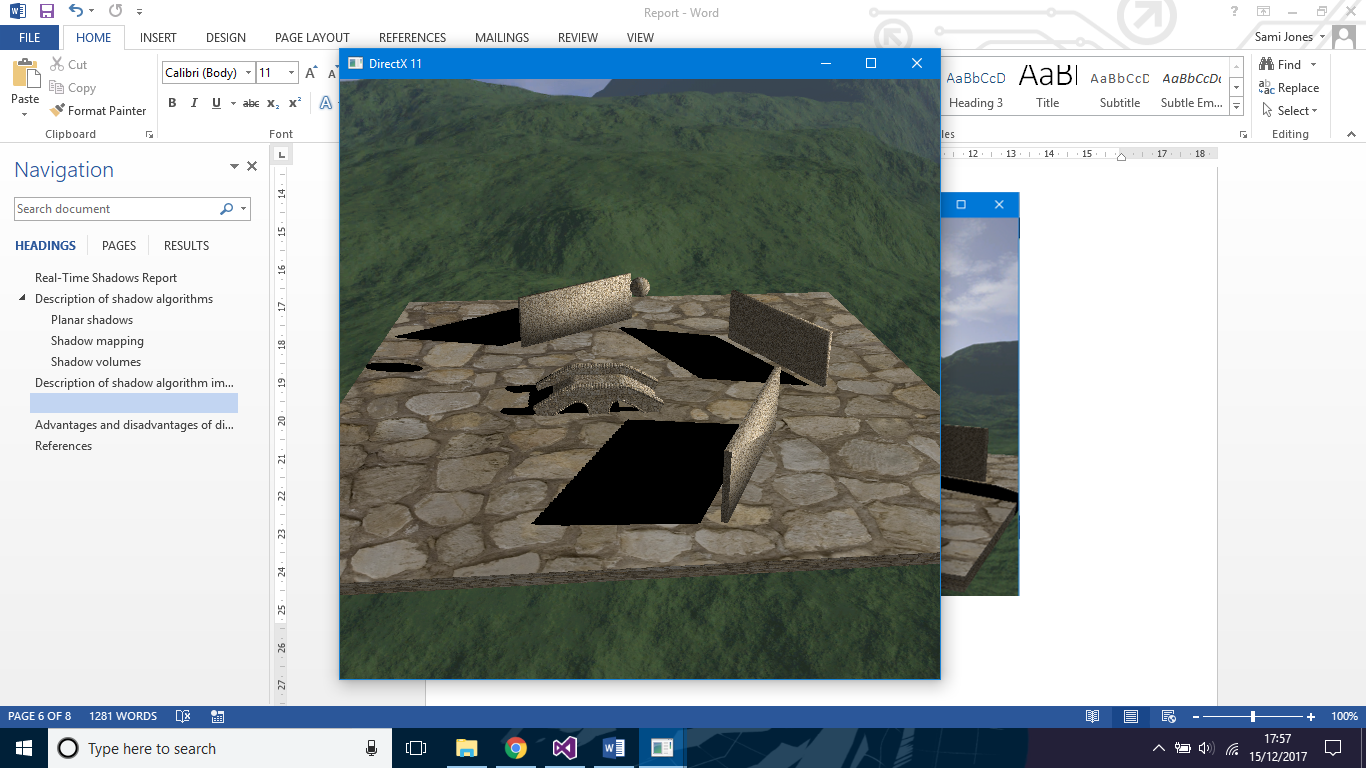


In the pixel shader, each pixel coordinate (transformed to the light camera’s perspective in the vertex shader), is compared with the corresponding pixel in the shadow map. A pixel will only be sampled from the shadow map if the pixel input to the shader lies within the width and height bounds of the shadow map texture. If the z-coordinate (depth-coordinate) of the input pixel is less than the depth of the shadow map at that pixel, then the pixel is visible to the light and can be rendered as normal. If the z-coordinate of the input pixel is more than the depth of the shadow map at that pixel, then the pixel is concealed by another object, is not visible to the light, and can be rendered in black. See below.



The output of the shadow mapping algorithm can be seen below.





# Advantages and disadvantages of different shadow algorithms

Williams (1978) states a number of advantages for the shadow mapping algorithm. The first is that objects in the scene do not need to be sorted in order for the algorithm to work correctly, so the algorithm is useful for rendering complex scenes with many objects. The second advantage is that the algorithm is efficient; it scales linearly as the depth complexity of the scene increases. Williams states that in general, the rendering cost of the algorithm is roughly double that of rendering the scene normally (but not quite double, since there is no need to shade pixels or render to the screen when rendering to the depth buffer from the light’s perspective). Shadow mapping is known to be more efficient than shadow volumes.

According to Smart (2015), another advantage of the shadow mapping algorithm is that it is flexible; performance can be adjusted by changing the resolution of the shadow map. A low-resolution shadow map will be faster to create, but the shadow quality will be reduced. A high-resolution map will be slower to create, but better quality. Shadow volumes do not allow the user this flexibility. Also, shadow mapping is based around rendering a depth map to a texture. This means that the shadows can be modified in various ways, for example by making them textured, or partially transparent. This is more difficult to achieve when using shadow volumes.

One of the main disadvantages of shadow mapping is that it often produces aliasing called shadow acne (Luna, 2012). Due to the limited resolution and accuracy of the shadow map, the edges of shadows are often jagged, especially when the light source is far away. Another disadvantage is that the algorithm requires an additional render pass for every additional shadow-casting light.

According to Kolivand and Sunar (2011), shadow mapping is not ideal for situations where a point light is placed in a central position relative to objects. The reason for this is that a shadow map would then need to be rendered for every direction relative to the light, as opposed to the one normally required when the light is placed some distance away from the main scene.

According to Tsiombikas (no date), one of the benefits of shadow volumes is that it can be applied to as many lights and objects as needed without too much difficulty in implementation. Adding a light simply requires that more shadow volumes be built in the first step of the algorithm. Another benefit is that shadow volumes produces shadows with more precise and accurate shapes than shadow mapping. This is because the algorithm is based on the actual geometry of the shadow-casting objects. Another benefit is that it does not matter where the light is placed, since the algorithm does not rely on objects being in particular positions relative to the light.

Tsiombikas (no date) also states some of the disadvantages of shadow volumes. The main one is that shadow volumes is very CPU-intensive, especially when the objects casting shadows have many polygons. In addition, the algorithm is only capable of creating shadows with hard edges. Shadow volumes also has difficulty rendering transparent shadows. For example, in the case of a textured quad with transparency, the shadow will be rectangular, as it is based on the geometry of the object.

# References

Crow, F. (1977) ‘Shadow algorithms for computer graphics’, *ACM SIGGRAPH Computer Graphics*, 11(2), pp. 242-248.

Kainulainen, J. (2004) *An Introduction to Stencil Shadow Volumes.* Available at: https://web.archive.org/web/20070208131226/http://www.devmaster.net/articles/shadow\_volumes/ (Accessed: 15 December 2017).

Kolivand, H. and Sunar, M. (2011) ‘Shadow mapping or shadow volume?’, *International Journal of New Computer Architectures and their Applications,* 1(2), pp. 275-281.

Luna, F. (2012) *Introduction to 3D game programming with DirectX 11.* Dulles, Va.: Mercury Learning and Information.

Smart, J. (2015) *Why is shadow mapping the standard?* Available at: https://gamedev.stackexchange.com/questions/106676/why-is-shadow-mapping-the-standard (Accessed: 15 December 2017).

Tsiombikas, J. (No date) *Volume Shadows Tutorial.* Available at: http://nuclear.mutantstargoat.com/articles/volume\_shadows\_tutorial\_nuclear.pdf (Accessed: 15 December 2017).

Williams, L. (1978) ‘Casting curved shadows on curved surfaces’, *ACM SIGGRAPH Computer Graphics,* 12(3), pp. 270-274.