Team: J.J. Abrahams

Members: Benjamin Johnson and Mitchell Marshe

**Manual**

Compiling:

cd into the project directory

mkdir build

cd build

cmake ..

make -j8

Running:

cd into the project directory

cd build

./bin/lens

Controls:

ESC - exit.

W - move or zoom forward.

A - strafe or rotate towards the left.

S - move or zoom backward.

D - strafe or rotate towards the right.

Up Arrow - move camera upward.

Left Arrow - roll camera towards the left.

Down Arrow - move camera downward.

Right Arrow - roll camera towards the right.

Left Mouse - dynamic local or center camera rotation.

Right Mouse - dynamic camera zoom.

C - toggle between orbital or first person camera mode.

Q - decrease exposure setting.

E - increase exposure setting.

Space - Take a photo in the game. Take photos to gain points.

J - snap a screenshot. Kept in screenshots folder.

1 - Menger cube first level fractile.

2 - Menger cube second level fractile.

3 - toggle to show mesh objects.

4 - toggle to show lens effects.

Bonus:

Inside main.cc, there are the following variable that one can play around with:

drunkMode - enable a blurry vision of drunkenness.

light\_rays\_for\_bokeh - increase number for depth of field. Doesn’t really work well though.

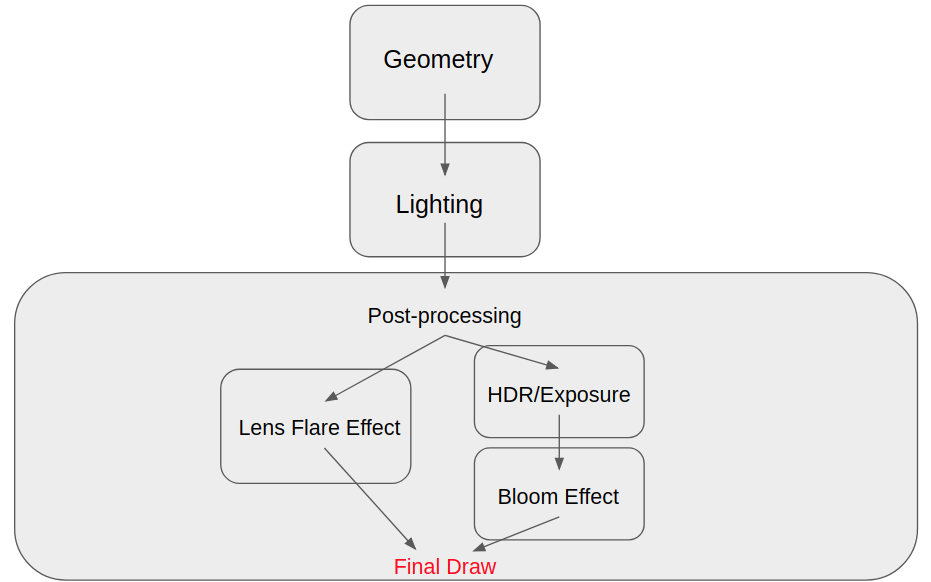
showGui - toggles the gui on/off.

Final Graphics Project Report

Background:

This final graphics project focused on using opengl and glsl to implement a post-processing shader pipeline and post-processing shader effects. This project is built on top of the menger assignment we did in class. For reference, the menger assignment did not have a post-processing shader pipeline and simply used the single default opengl framebuffer to render basic geometry.

The post-processing shader pipeline is implemented by rendering all geometry in the scene to a default textures color buffer object which can then be processed through shaders which take the texture data, apply effects, and output a new textures. The final culmination of textures which have had effects applied to them is combined in a final default shader which simply adds up the values of all the input textures and then writes the output to a final texture which is drawn to a screen space quad which the user can see. This process of stacking and combining various shaders allows for a pipeline which is easy to extend and develop new effects with.



Above is an example of how the pipeline looks with the lens flare, high dynamic range, and light bloom effects applied. This chart visualizes how our geometry is rendered, lighting is applied, and the result of this is then output to a texture which is then used to process other effects that are layered on top of the original geometry image for the final screen draw. This pipeline technique, and many of the shaders we developed to exemplify the pipeline, are inspired by the tutorials on [www.learnopengl.com](http://www.learnopengl.com).

By implementing this pipeline, we have created a system which allows us to experiment with various post-processing shaders without having to deal as much with opengl buffers and state, which will allow us to easily build on and extend the game with more shaders in the future. By comparison, when we started, menger did not have these features and we had to dig around in the weeds for a while in order to establish a solid post-processing pipeline for adding these shaders.

The post processing effects we are applying are simulating common camera lens effects such as lens flare and depth of field. A lens flare is the phenomenon of bright lights being reflected inside of a camera lens so that “ghost” artifacts and strange light bounces are created. Depth of field is the phenomenon of objects being at various levels of sharpness and blur depending on whether they are in the camera’s plane of focus. We simulate this by blurring and duplicating vertices so that objects “bloom” in and out of focus.

What we have accomplished:

Our final project is to implement different shaders for camera lens effects. These various shaders are applied in the post-processing phase of the rendering pipeline. The following lens effects that we have currently developed are lens flare, screen blurring, brightness, exposure, HDR, down sampling, and depth of field (sort of). These effects are utilized in our Photography Simulator demo, where a user snaps awesome photos over specific objects for points.

Lens Flare - when a bright source of light casts down into a camera lens at a certain angle, the mirrors inside the camera causes ghosts or repetitions of whatever is on the scene. For instance, if the camera is pointed at bright cube at some angle, chances are that cube will appear 2 or 3 times in a linear perspective line with a ghost-like effect. We also apply a radial color texture to add more interesting colors to the lens flare. We also apply a halo effect around the edge of the screen to simulate the subtle fish eye reflection effect that can be scene in many lens flares. This technique is based on this article: <http://john-chapman-graphics.blogspot.com/2013/02/pseudo-lens-flare.html>

Color Picking - a technique to identify objects based on unique colors as identification. The applied color to the object is invisible to the user because a color buffer is utilized.

Gaussian Screen Blurring - the effect of mixing in pixels to get rid of hard edges, causing a sort of smoothness to the imagery.

Brightness - the blooming effect around light sources. Otherwise, known as a glow.

Exposure - a control for how much white or black is in the scene.

HDR - high dynamic range allows for adjustment of exposure which can bring out details in very bright or very dark spots in the image.

Downsampling - reduces the screen image resolution and brings out the bright spots for applying lens flare or bloom effects.

Depth of Field - an effect where the foreground is in focus while the background is blurred out. This effect can also be applied in the foreground. Currently the effect works for blowing out the objects by adjusting the light\_rays\_for\_bokeh, but the blurring effect which follows does not create a realistic looking depth of field effect.

References:

* <http://john-chapman-graphics.blogspot.com/2013/02/pseudo-lens-flare.html>
* <http://www.inass.org/share/20100106175619849.pdf>
* <https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch23.html>
* <https://en.wikibooks.org/wiki/OpenGL_Programming/Depth_of_Field>
* <https://learnopengl.com/>
* <http://www.opengl-tutorial.org/miscellaneous/clicking-on-objects/picking-with-an-opengl-hack/>