Intermediate Computer Graphics

Assignment #2

INFR 2350

Dr. Hogue

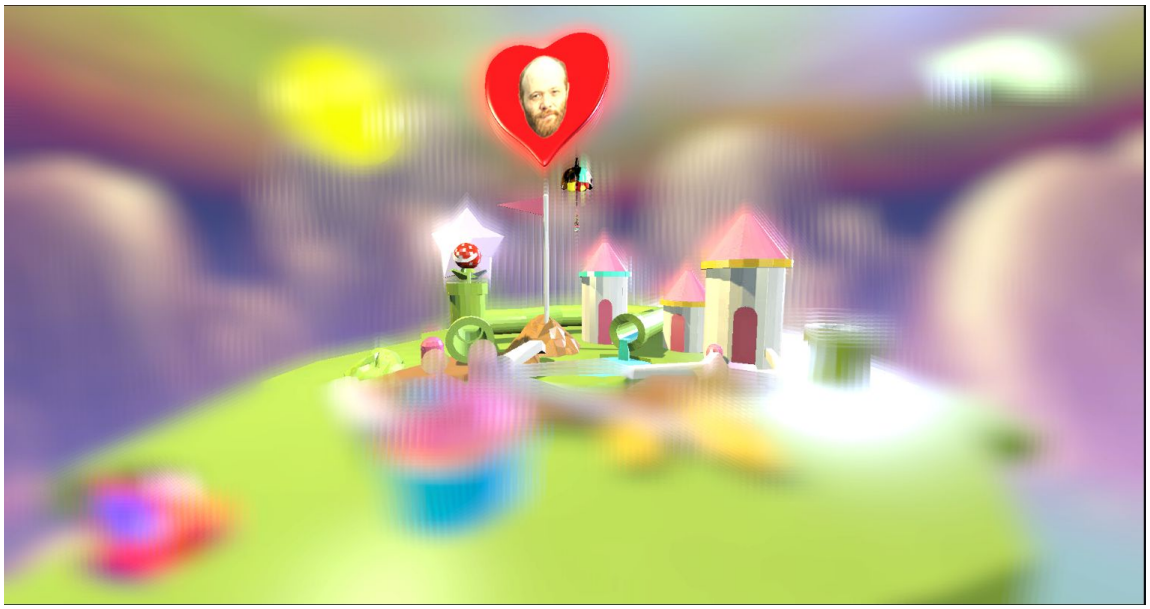
Safa Nazir - 100654328

Lillian Fan - 100672027

PART 1: Simple Depth of Field

Description of Math:

Our “simple” depth of field algorithm will take in the focal distance, near and far distances and will interpolate the colour of the pixel between a blurred version of the scene and the in-focus part of the scene.
Pictured below is the simple DOF implemented.



```
void main()
{
    vec2 texOffset = gl_FragCoord.xy * uPixelSize;
    float aspectRatio = uResolution.x/uResolution.y;

    float depth = texture(uTexDepth, texOffset).r;
    float factor = abs(depth - uFocus);

    vec2 dofblur = vec2(clamp((factor * uBias - uThreshold), 0.0, uBlurClamp),
                        clamp((factor * uBias - 0.01), 0.0, uBlurClamp)) * aspectRatio;

    vec4 col = vec4(0, 0, 0, 1);

    col.rgb += texture(uTexScene, texOffset + vec2(0.0, - 4.0 * dofblur.x)).rgb * 0.06;
    col.rgb += texture(uTexScene, texOffset + vec2(0.0, - 3.0 * dofblur.x)).rgb * 0.09;
    col.rgb += texture(uTexScene, texOffset + vec2(0.0, - 2.0 * dofblur.x)).rgb * 0.12;
    col.rgb += texture(uTexScene, texOffset + vec2(0.0, -       dofblur.x)).rgb * 0.15;
    col.rgb += texture(uTexScene, texOffset                                ).rgb * 0.16;
    col.rgb += texture(uTexScene, texOffset + vec2(0.0,         dofblur.y)).rgb * 0.15;
    col.rgb += texture(uTexScene, texOffset + vec2(0.0,   2.0 * dofblur.y)).rgb * 0.12;
    col.rgb += texture(uTexScene, texOffset + vec2(0.0,   3.0 * dofblur.y)).rgb * 0.09;
    col.rgb += texture(uTexScene, texOffset + vec2(0.0,   4.0 * dofblur.y)).rgb * 0.06;

    outColor = col;
}
```

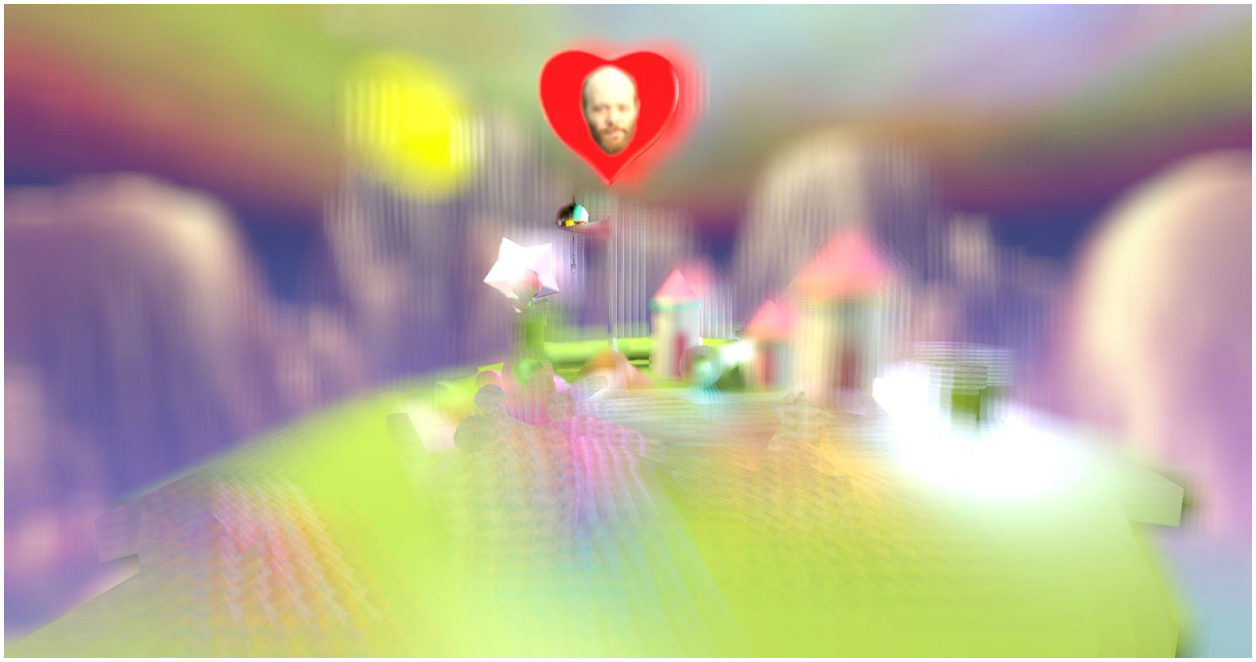
Above is the shader code for blurring the horizontal/X-axis.

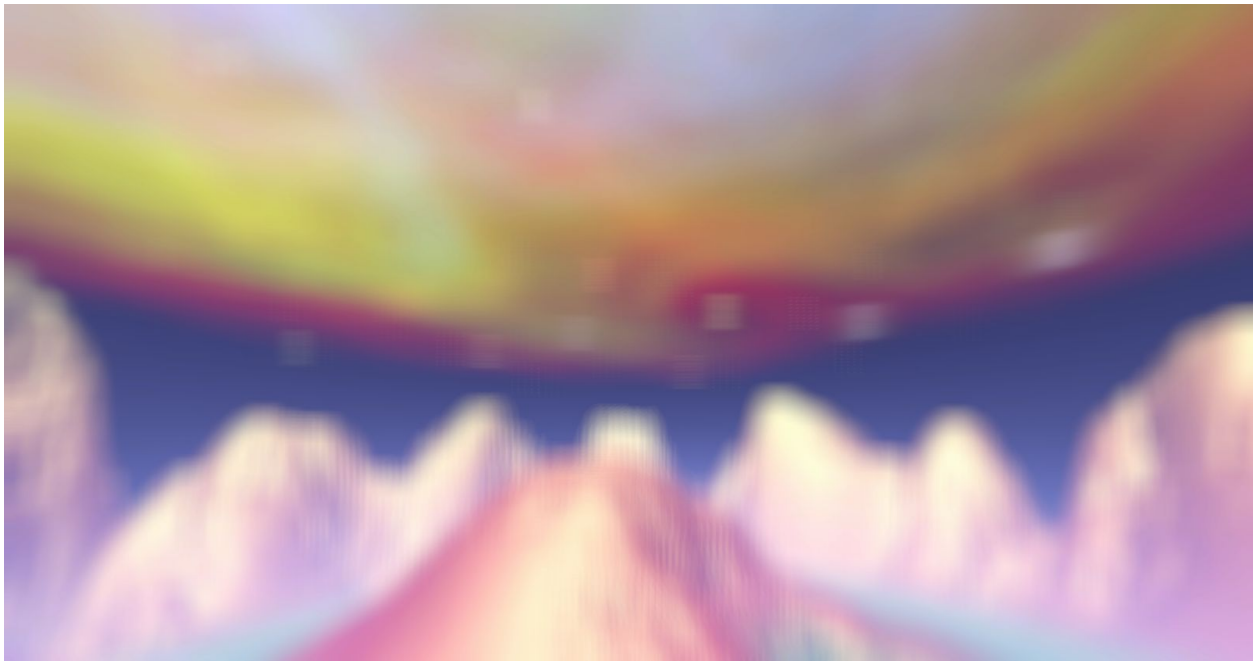
```
void main()
{
    vec2 texOffset = gl_FragCoord.xy * uPixelSize;
    float aspectRatio = uResolution.x/uResolution.y;

    float depth = texture(uTexDepth, texOffset).r;
    float factor = abs(depth - uFocus);


    vec2 dofblur = vec2(clamp((factor * uBias - uThreshold), 0.0, uBlurClamp),
                        clamp((factor * uBias - 0.01), 0.0, uBlurClamp)) * aspectRatio;

    vec4 col = vec4(0, 0, 0, 1);

    col.rgb += texture(uTexScene, texOffset + vec2(- 4.0 * dofblur.x, 0.0)).rgb * 0.06;
    col.rgb += texture(uTexScene, texOffset + vec2(- 3.0 * dofblur.x, 0.0)).rgb * 0.09;
    col.rgb += texture(uTexScene, texOffset + vec2(- 2.0 * dofblur.x, 0.0)).rgb * 0.12;
    col.rgb += texture(uTexScene, texOffset + vec2(-       dofblur.x, 0.0)).rgb * 0.15;
    col.rgb += texture(uTexScene, texOffset                               ).rgb * 0.16;
    col.rgb += texture(uTexScene, texOffset + vec2(        dofblur.y, 0.0)).rgb * 0.15;
    col.rgb += texture(uTexScene, texOffset + vec2(  2.0 * dofblur.y, 0.0)).rgb * 0.12;
    col.rgb += texture(uTexScene, texOffset + vec2(  3.0 * dofblur.y, 0.0)).rgb * 0.09;
    col.rgb += texture(uTexScene, texOffset + vec2(  4.0 * dofblur.y, 0.0)).rgb * 0.06;

    outColor = col;
}
```

Above is the shader code for blurring the vertical/Y-axis.

They are the simple depth of field effect that we covered in one of our tutorials. The DOF effect will affect every pixel on screen and shift the pixels on both x and y axis.

PART 2: Improved Depth of Field





Pictured above is the improved depth of field implemented in the engine. It might be a bit difficult to see the square shape but it is obvious when we move the camera towards the sky. The sky box includes stars that are basically direct lights so it's easier to see the bokeh effect.

Description of Math:

To create the improved depth of field we have to utilize the circle of confusion which creates a more realistic blur. The equation we used is pictured below:

$$CoC(D) = |A * \frac{F * (P - D)}{D * (P - F)}|$$

Where A = Aperture
F = Focal Length
P = Plane in focus
D = Object distance
I = Image distance

$$\frac{1}{P} + \frac{1}{I} = \frac{1}{F}$$

```
vec2 texOffset = gl_FragCoord.xy * uPixelSize;

float depth = texture(uTexDepth, texOffset).r;

float myCoc = abs(uBias * uFocalLength * (uFocus - depth)  / (depth * (uFocus - uFocalLength)));
myCoc = clamp(myCoc, 0.0, uBlurClamp);
```

Pictured above is the CoC(D) formula implemented in our shader.

```
vec2 points[50] = {
    vec2(-0.9, 0.9),
    vec2(-0.7, 0.9),
    vec2(-0.5, 0.9),
    vec2(-0.3, 0.9),
    vec2(-0.1, 0.9),

    vec2(-0.9, 0.7),
    vec2(-0.7, 0.7),
    vec2(-0.5, 0.7),
    vec2(-0.3, 0.7),
    vec2(-0.1, 0.7),

    vec2(-0.9, 0.5),
    vec2(-0.7, 0.5),
    vec2(-0.5, 0.5),
    vec2(-0.3, 0.5),
    vec2(-0.1, 0.5),

    vec2(-0.9, 0.3),
    vec2(-0.7, 0.3),
```

Above are all the coordinates that were used to create a final result of a square-looking shape.

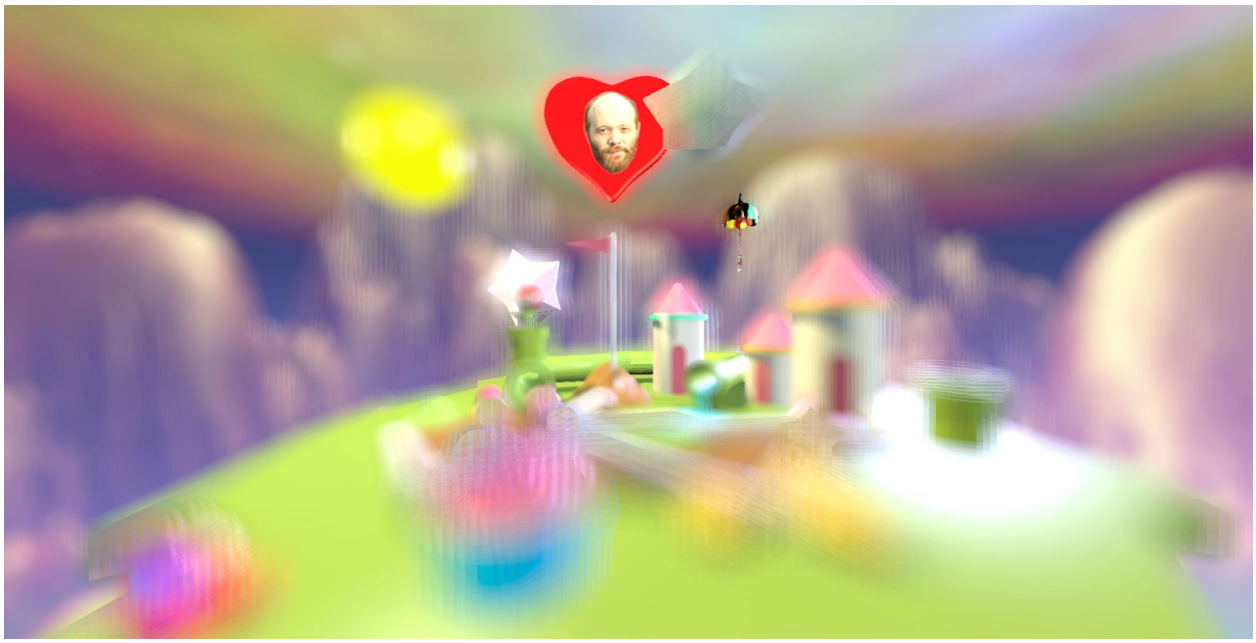
```
for(int i=0; i<50; i++)
{
    vec2 myPoint = points[i];
    myPoint *= myCoc;
    myColor += texture(uTexScene, texOffset + myPoint).rgb;
}

myColor *= 1.0 / 50;

outColor = vec4(myColor, 1.0);
```

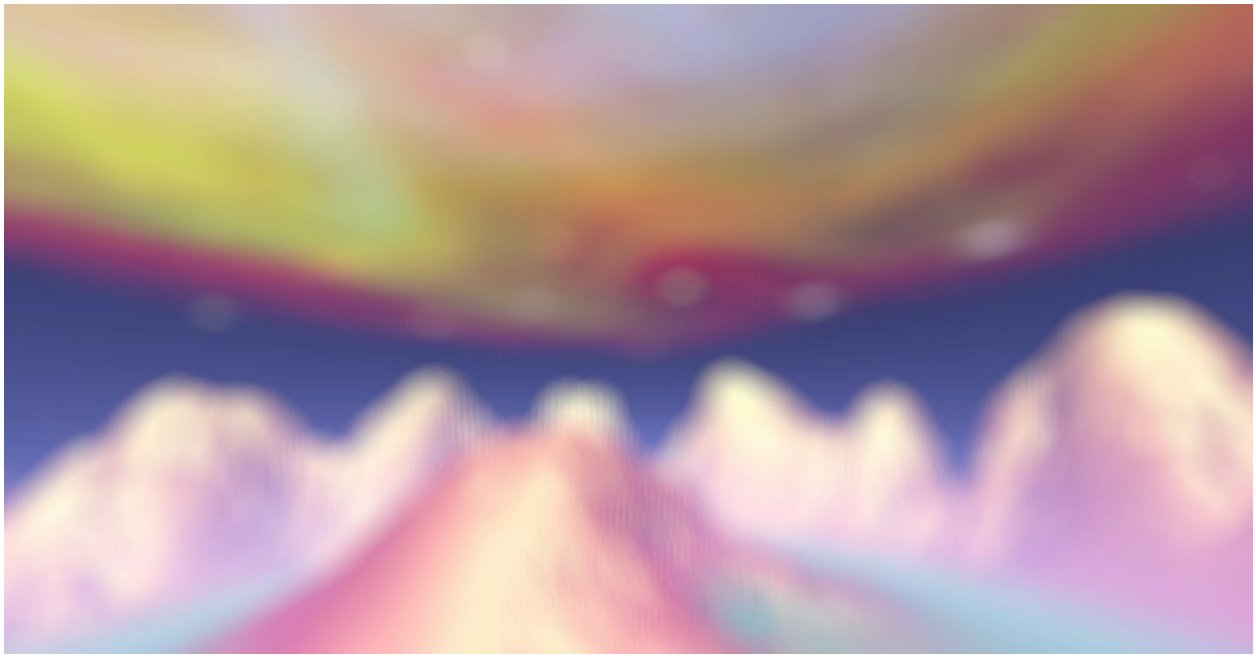
Finally, the CoC formula is used along with the coordinates of the square shape and the original texture which will generate the blurring effect.

When we tried to make the square bokeh effect work, we found that the square turned into a rectangle shape. And when we tried with a rectangle shape, it resulted in the square effect we wanted.

PART 3: Bokeh Sampling and Performance

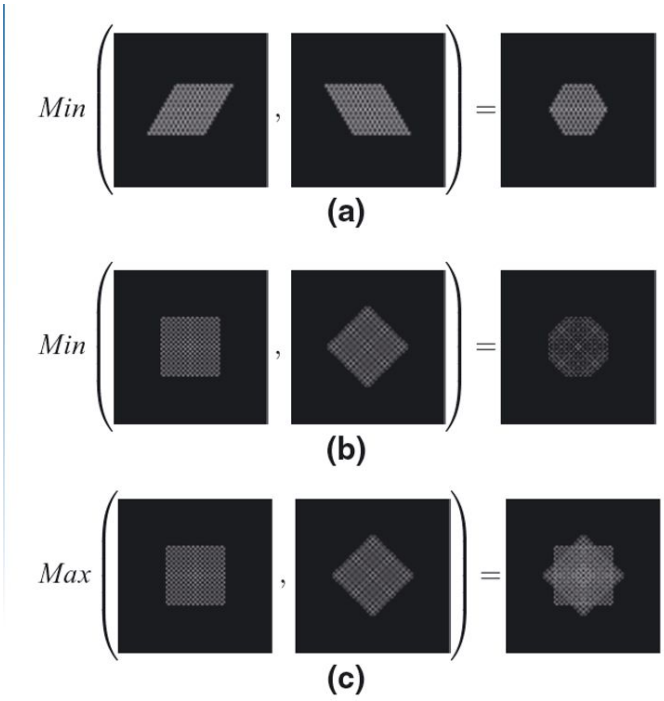
Circle Bokeh:



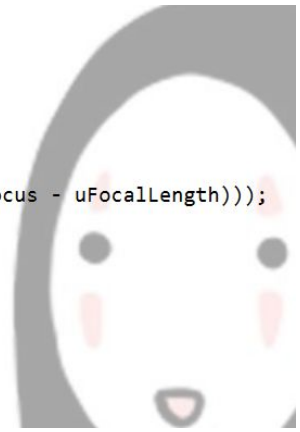


Pictured above is the circle bokeh effect.

Ideally, a bokeh effect is created by shifting the pixels in one direction and another and adding them together which will result in the desired shape, as seen below.



We found it difficult to implement so we were told by the professor to manually create the shape of the bokeh effect to use in the shader. This was done by plotting a shape on a grid and using the coordinates of each point to create a full shape. This was used with the same CoC function that was implemented in Part 2, which blurs our scene.

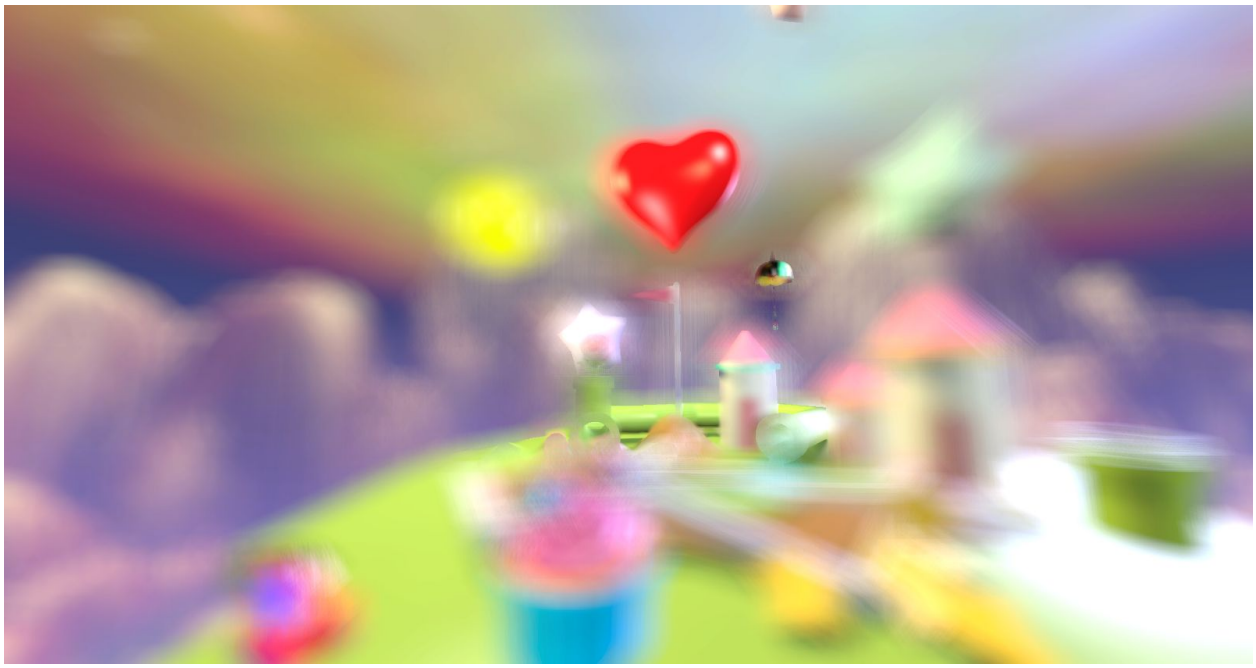


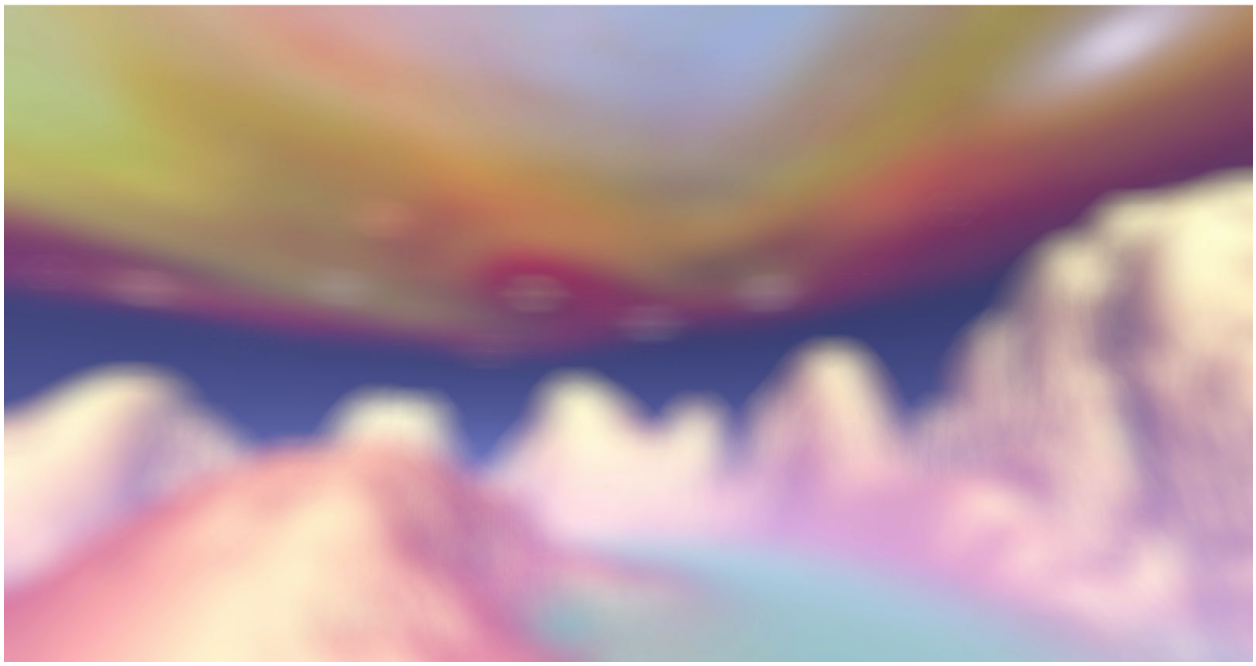
```
void main()
{
    vec2 texOffset = gl_FragCoord.xy * uPixelSize;

    float depth = texture(uTexDepth, texOffset).r;

    float myCoc = abs(uBias * uFocalLength * (uFocus - depth)  / (depth * (uFocus - uFocalLength)));
    myCoc = clamp(myCoc, 0.0, uBlurClamp);

    vec3 myColor = vec3(0, 0, 0);

    vec2 points[61] = {
        vec2(-0.2, 1),
        vec2(   0, 1),
        vec2( 0.2, 1)
```

```
    for(int t=0; t<61; t++)
    {
        vec2 myPoint = points[t];
        myPoint *= myCoc;
        myColor += texture(uTexScene, texOffset + myPoint).rgb;
    }

    myColor.rgb *= 1.0 / 61;

    outColor = vec4(myColor, 1.0);


}
```

Hexagonal Bokeh:

Below is a snippet of the shader we used to implement the hexagonal bokeh effect:





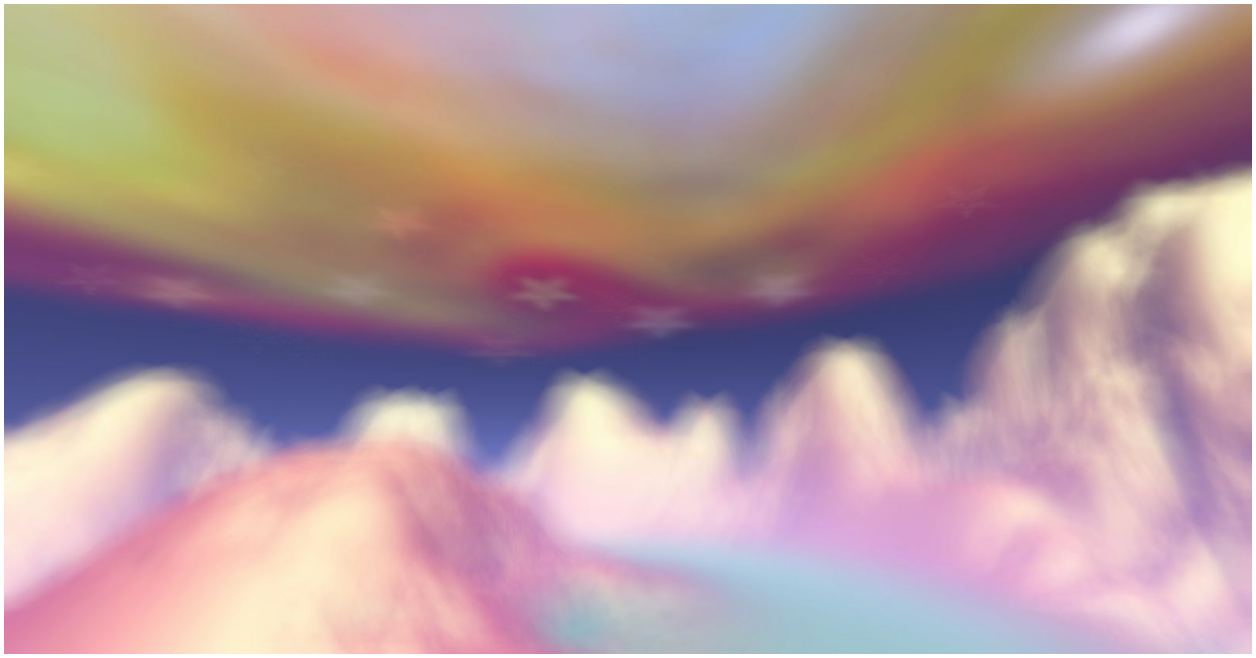
For the hexagonal shader we followed the same approach: Create the hexagon shape with several coordinates and use the CoC formula to create the resulting scene.

```
void main()
{
    vec2 texOffset = gl_FragCoord.xy * uPixelSize;

    float depth = texture(uTexDepth, texOffset).r;

    float myCoc = abs(uBias * uFocalLength * (uFocus - depth)  / (depth * (uFocus - uFocalLength)));
    myCoc = clamp(myCoc, 0.0, uBlurClamp);

    vec3 myColor  = vec3(0, 0, 0);

    vec2 points[180] = {
        vec2(-0.04, 0.62),
        vec2(-0.12, 0.62),
        vec2(-0.18, 0.62),
        vec2(-0.25, 0.62),
        vec2(-0.32, 0.62),
        vec2(-0.04, 0.54),
```

```
    for(int t=0; t<180; t++)
    {
        vec2 myPoint = points[t];
        myPoint *= myCoc;
        myColor  += texture(uTexScene, texOffset + myPoint).rgb;
    }

    myColor .rgb *= 1.0 / 180;

    outColor = vec4(myColor , 1.0);
}
```

Star Bokeh:

Below is what our scene looks like with the star bokeh effect:



The star shader was created the same way as the other bokeh effects - a star shape was created using vec2 coordinates and was used with the CoC formula which results in star bokehs all around the scene.

```
void main()
{
    vec2 texOffset = gl_FragCoord.xy * uPixelSize;

    float depth = texture(uTexDepth, texOffset).r;

    float myCoc = abs(uBias * uFocalLength * (uFocus - depth)  / (depth * (uFocus - uFocalLength)));
    myCoc = clamp(myCoc, 0.0, uBlurClamp);

    vec3 myColor = vec3(0, 0, 0);

    vec2 points[108] = {
        vec2(0, 0.6),
        vec2(-0.03, 0.56),
        vec2(-0.07, 0.472),
        vec2(-0.01, 0.42),
        vec2(0.39,-0.57),
    };

    for(int i=0; i<108; i++)
    {
        vec2 myPoint = points[i];
        myPoint *= myCoc;
        myColor += texture(uTexScene, texOffset + myPoint).rgb;
    }

    myColor.rgb *= 1.0 / 108;

    outColor = vec4(myColor, 1.0);


}
```

Above is the shader code for the star bokeh effect. The only thing different from the other bokeh effects is the position of the coordinates.

Sources:

lecture_DOF.ppt, Dr Hogue.

http://ivizlab.sfu.ca/papers/cgf2012.pdf