CPI411 Graphics for Games
Final Lab (Glow Shader for Tron like effect)

For this lab we will need four files added to the solution

- FinalLab.cs
- ScreenRenderer.cs
- GlowFilter.cs
- Glow.fx

A. First Concept:

In this lab you will make a basic glow shader that looks for the brightest alpha in a photo to specify its glow point to apply the shader effect too. You will then add a blur effect to give it the sleek Tron inspired look from any picture that you put through the engine. It will naturally highlight the brightest points of the photo then you can adjust the strength of the effect or lower the threshold to have more of the image glow. This effect can really make an image come to life and is a good shader to have in a game to highlight things without being intrusive or ugly.

B. Glow Shader:

First you need to create your variables:

```
float2 InverseResolution;
float Threshold = 0.8f;
float Radius;
float Strength;
float StreakLength = 1;
Texture2D ScreenTexture;
```

Create sampler state:

```
SamplerState LinearSampler
{
Texture = <ScreenTexture>;

MagFilter = LINEAR;
MinFilter = LINEAR;
Mipfilter = LINEAR;

AddressU = CLAMP;
AddressV = CLAMP;
};
```

Vertex Shader:

```
struct VertexShaderInput
{
	float3 Position : POSITION0;
	float2 TexCoord : TEXCOORD0;
};

struct VertexShaderOutput
{
	float4 Position : POSITION0;
	float2 TexCoord : TEXCOORD0;
};

// Vertex Shader
VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
{
	VertexShaderOutput output;
	output.Position = float4(input.Position, 1);
	output.TexCoord = input.TexCoord;
	return output;
}
```

Pixel Shaders to change effect:

```
//Extracts the pixels we want to blur
float4 ExtractPS(float4 pos : SV_POSITION, float2 texCoord : TEXCOORD0) : SV_TARGET0
{
	float4 color = ScreenTexture.Sample(LinearSampler, texCoord);

	float avg = (color.r + color.g + color.b) / 3;

	if (avg > Threshold)
	{
		return color * (avg - Threshold) / (1 - Threshold);// * (avg - Threshold);
	}

	return float4(0, 0, 0, 0);
}

//Extracts the pixels we want to blur, but considers luminance instead of average rgb
float4 ExtractLuminancePS(float4 pos : SV_POSITION, float2 texCoord : TEXCOORD0) :
SV_TARGET0
```

```
{
	float4 color = ScreenTexture.Sample(LinearSampler, texCoord);

	float luminance = color.r * 0.21f + color.g * 0.72f + color.b * 0.07f;

	if (luminance > Threshold)
	{
		return color * (luminance - Threshold) / (1 - Threshold);// *(luminance - 
Threshold);
		//return saturate((color - Threshold) / (1 - Threshold));
	}

	return float4(0, 0, 0, 0);
}

//Downsample to the next mip, blur in the process
float4 DownsamplePS(float4 pos : SV_POSITION, float2 texCoord : TEXCOORD0) : 
SV_TARGET0
{
	float2 offset = float2(StreakLength * InverseResolution.x, 1 * InverseResolution.y);

	float4 c0 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-2, -2) * offset);
	float4 c1 = ScreenTexture.Sample(LinearSampler, texCoord + float2(0,-2)*offset);
	float4 c2 = ScreenTexture.Sample(LinearSampler, texCoord + float2(2, -2) * offset);
	float4 c3 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-1, -1) * offset);
	float4 c4 = ScreenTexture.Sample(LinearSampler, texCoord + float2(1, -1) * offset);
	float4 c5 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-2, 0) * offset);
	float4 c6 = ScreenTexture.Sample(LinearSampler, texCoord);
	float4 c7 = ScreenTexture.Sample(LinearSampler, texCoord + float2(2, 0) * offset);
	float4 c8 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-1, 1) * offset);
	float4 c9 = ScreenTexture.Sample(LinearSampler, texCoord + float2(1, 1) * offset);
	float4 c10 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-2, 2) * offset);
	float4 c11 = ScreenTexture.Sample(LinearSampler, texCoord + float2(0, 2) * offset);
	float4 c12 = ScreenTexture.Sample(LinearSampler, texCoord + float2(2, 2) * offset);

	return Box4(c0, c1, c5, c6) * 0.125f +
	Box4(c1, c2, c6, c7) * 0.125f +
	Box4(c5, c6, c10, c11) * 0.125f +
	Box4(c6, c7, c11, c12) * 0.125f +
	Box4(c3, c4, c8, c9) * 0.5f;
```

```
}

//Upsample to the former MIP, blur in the process
float4 UpsamplePS(float4 pos : SV_POSITION, float2 texCoord : TEXCOORD0) :
SV_TARGET0
{
	float2 offset = float2(StreakLength * InverseResolution.x, 1 * InverseResolution.y) *
Radius;

	float4 c0 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-1, -1) * offset);
	float4 c1 = ScreenTexture.Sample(LinearSampler, texCoord + float2(0, -1) * offset);
	float4 c2 = ScreenTexture.Sample(LinearSampler, texCoord + float2(1, -1) * offset);
	float4 c3 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-1, 0) * offset);
	float4 c4 = ScreenTexture.Sample(LinearSampler, texCoord);
	float4 c5 = ScreenTexture.Sample(LinearSampler, texCoord + float2(1, 0) * offset);
	float4 c6 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-1,1) * offset);
	float4 c7 = ScreenTexture.Sample(LinearSampler, texCoord + float2(0, 1) * offset);
	float4 c8 = ScreenTexture.Sample(LinearSampler, texCoord + float2(1, 1) * offset);

	//Tentfilter  0.0625f
	return 0.0625f * (c0 + 2 * c1 + c2 + 2 * c3 + 4 * c4 + 2 * c5 + c6 + 2 * c7 + c8) *
Strength + float4(0, 0,0,0); //+ 0.5f * ScreenTexture.Sample(c_texture, texCoord);

}

//Upsample to the former MIP, blur in the process, change offset depending on luminance
float4 UpsampleLuminancePS(float4 pos : SV_POSITION, float2 texCoord : TEXCOORD0) :
SV_TARGET0
{
	float4 c4 = ScreenTexture.Sample(LinearSampler, texCoord);  //middle one

	/*float luminance = c4.r * 0.21f + c4.g * 0.72f + c4.b * 0.07f;
	luminance = max(luminance, 0.4f);
*/
	float2 offset = float2(StreakLength * InverseResolution.x, 1 * InverseResolution.y) *
Radius; /// luminance;

	float4 c0 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-1, -1) * offset);
	float4 c1 = ScreenTexture.Sample(LinearSampler, texCoord + float2(0, -1) * offset);
	float4 c2 = ScreenTexture.Sample(LinearSampler, texCoord + float2(1, -1) * offset);
```

```
	float4 c3 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-1, 0) * offset);
	float4 c5 = ScreenTexture.Sample(LinearSampler, texCoord + float2(1, 0) * offset);
	float4 c6 = ScreenTexture.Sample(LinearSampler, texCoord + float2(-1, 1) * offset);
	float4 c7 = ScreenTexture.Sample(LinearSampler, texCoord + float2(0, 1) * offset);
	float4 c8 = ScreenTexture.Sample(LinearSampler, texCoord + float2(1, 1) * offset);

	return 0.0625f * (c0 + 2 * c1 + c2 + 2 * c3 + 4 * c4 + 2 * c5 + c6 + 2 * c7 + c8) *
Strength + float4(0, 0, 0, 0); //+ 0.5f * ScreenTexture.Sample(c_texture, texCoord);

}
```

Pass each technique through with their own Pixel Shader:

```
technique Extract
{
	pass Pass1
	{
		VertexShader = compile vs_4_0 VertexShaderFunction();
		PixelShader = compile ps_4_0 ExtractPS();
	}
}

technique ExtractLuminance
{
	pass Pass1
	{
		VertexShader = compile vs_4_0 VertexShaderFunction();
		PixelShader = compile ps_4_0 ExtractLuminancePS();
	}
}

technique Downsample
{
	pass Pass1
	{
		VertexShader = compile vs_4_0 VertexShaderFunction();
		PixelShader = compile ps_4_0 DownsamplePS();
	}
}

technique Upsample
```

```
{
        pass Pass1
        {
                VertexShader = compile vs_4_0 VertexShaderFunction();
                PixelShader = compile ps_4_0 UpsamplePS();
        }
}


technique UpsampleLuminance
{
        pass Pass1
        {
                VertexShader = compile vs_4_0 VertexShaderFunction();
                PixelShader = compile ps_4_0 UpsampleLuminancePS();
        }
}
```

C. Glow Filter:

Add these using statements:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
```

All variables for the glow effect to take place on top of original image:

```
      //Resolution
       private int width;
       private int height;

       //RenderTargets
       private RenderTarget2D glowRenderTarget2D0;
       private RenderTarget2D glowRenderTarget2D1;
       private RenderTarget2D glowRenderTarget2D2;
       private RenderTarget2D glowRenderTarget2D3;
       private RenderTarget2D glowRenderTarget2D4;
       private RenderTarget2D glowRenderTarget2D5;

       private SurfaceFormat renderTargetFormat;

       //Objects
```

```
private GraphicsDevice graphicsDevice;
private ScreenRenderer screenRenderer;

//Shader + variables
private Effect glowEffect;

private EffectPass glowPassExtract;
private EffectPass glowPassExtractLuminance;
private EffectPass glowPassDownsample;
private EffectPass glowPassUpsample;
private EffectPass glowPassUpsampleLuminance;

private EffectParameter glowParameterScreenTexture;
private EffectParameter glowInverseResolutionParameter;
private EffectParameter glowRadiusParameter;
private EffectParameter glowStrengthParameter;
private EffectParameter glowStreakLengthParameter;
private EffectParameter glowThresholdParameter;

//Preset variables for different mip levels
private float glowRadius1 = 1.0f;
private float glowRadius2 = 1.0f;
private float glowRadius3 = 1.0f;
private float glowRadius4 = 1.0f;
private float glowRadius5 = 1.0f;

private float glowStrength1 = 1.0f;
private float glowStrength2 = 1.0f;
private float glowStrength3 = 1.0f;
private float glowStrength4 = 1.0f;
private float glowStrength5 = 1.0f;

public float glowStrengthMultiplier = 1.0f;

private float glowRadiusMultiplier = 1.0f;

public bool glowUseLuminance = true;
public int glowDownsamplePasses = 5;
```

Set up enum for all glow presets

```
//enums
public enum GlowPresets
{
   Wide,
   Focussed,
   Small,
   SuperWide,
   Cheap,
   One
};
```

Set up the functions for the different attributes of the effect:

```
 public GlowPresets GlowPreset
{
   get { return glowPreset; }
   set
   {
      if (glowPreset == value) return;

      glowPreset = value;
      SetGlowPreset(glowPreset);
   }
}
private GlowPresets glowPreset;


private Texture2D GlowScreenTexture { set {
glowParameterScreenTexture.SetValue(value); } }
private Vector2 GlowInverseResolution
{
   get { return glowInverseResolutionField; }
   set
   {
      if (value != glowInverseResolutionField)
      {
         glowInverseResolutionField = value;
         glowInverseResolutionParameter.SetValue(glowInverseResolutionField);
      }
   }
}
private Vector2 glowInverseResolutionField;
```

```
private float GlowRadius
{
   get
   {
      return glowRadius;
   }

   set
   {
      if (Math.Abs(glowRadius - value) > 0.001f)
      {
         glowRadius = value;
         glowRadiusParameter.SetValue(glowRadius * glowRadiusMultiplier);
      }

   }
}
private float glowRadius;

private float GlowStrength
{
   get { return glowStrength; }
   set
   {
      if (Math.Abs(glowStrength - value) > 0.001f)
      {
         glowStrength = value;
         glowStrengthParameter.SetValue(glowStrength * glowStrengthMultiplier);
      }

   }
}
private float glowStrength;

public float GlowStreakLength
{
   get { return glowStreakLength; }
   set
   {
```

```
            if (Math.Abs(glowStreakLength - value) > 0.001f)
            {
                glowStreakLength = value;
                glowStreakLengthParameter.SetValue(glowStreakLength);
            }
        }
    }
    private float glowStreakLength;

    public float GlowThreshold
    {
        get { return glowThreshold; }
        set
        {
            if (Math.Abs(glowThreshold - value) > 0.001f)
            {
                glowThreshold = value;
                glowThresholdParameter.SetValue(glowThreshold);
            }
        }
    }
    private float glowThreshold;
```

Load in all parameters of the effect and the screen:

```
        public void Load(GraphicsDevice graphicsDevice, ContentManager content, int
width, int height, SurfaceFormat renderTargetFormat = SurfaceFormat.Color,
ScreenRenderer screenrenderer = null)
    {
        this.graphicsDevice = graphicsDevice;
        UpdateResolution(width, height);

        this.screenRenderer = screenrenderer ?? new ScreenRenderer(graphicsDevice);

        this.renderTargetFormat = renderTargetFormat;

        //Load the shader parameters
        glowEffect = content.Load<Effect>("Glow");
        glowInverseResolutionParameter = glowEffect.Parameters["InverseResolution"];
        glowRadiusParameter = glowEffect.Parameters["Radius"];
        glowStrengthParameter = glowEffect.Parameters["Strength"];
```

```
            glowStreakLengthParameter = glowEffect.Parameters["StreakLength"];
            glowThresholdParameter = glowEffect.Parameters["Threshold"];

            glowParameterScreenTexture = glowEffect.Parameters["ScreenTexture"];

            glowPassExtract = glowEffect.Techniques["Extract"].Passes[0];
            glowPassExtractLuminance =
glowEffect.Techniques["ExtractLuminance"].Passes[0];
            glowPassDownsample = glowEffect.Techniques["Downsample"].Passes[0];
            glowPassUpsample = glowEffect.Techniques["Upsample"].Passes[0];
            glowPassUpsampleLuminance =
glowEffect.Techniques["UpsampleLuminance"].Passes[0];

//Default threshold.
            GlowThreshold = 0.8f;
            SetGlowPreset(GlowPreset);
```

Initialize the presets for the glow effect here is 2 of the 5 created in our enum(finish focussed, small, cheap, and one:

```
        private void SetGlowPreset(GlowPresets preset)
        {
            switch (preset)
            {
                case GlowPresets.Wide:
                    {
                        glowStrength1 = 0.5f;
                        glowStrength2 = 1;
                        glowStrength3 = 2;
                        glowStrength4 = 1;
                        glowStrength5 = 2;
                        glowRadius5 = 4.0f;
                        glowRadius4 = 4.0f;
                        glowRadius3 = 2.0f;
                        glowRadius2 = 2.0f;
                        glowRadius1 = 1.0f;
                        GlowStreakLength = 1;
                        glowDownsamplePasses = 5;
                        break;
                    }
                case GlowPresets.SuperWide:
```

```
            {
                glowStrength1 = 0.9f;
                glowStrength2 = 1;
                glowStrength3 = 1;
                glowStrength4 = 2;
                glowStrength5 = 6;
                glowRadius5 = 4.0f;
                glowRadius4 = 2.0f;
                glowRadius3 = 2.0f;
                glowRadius2 = 2.0f;
                glowRadius1 = 2.0f;
                GlowStreakLength = 1;
                glowDownsamplePasses = 5;
                break;
            }
```

Draw the filter over the image and apply effect:

```
public Texture2D Draw(Texture2D inputTexture, int width, int height)
    {
        //Check if we are initialized
        if (graphicsDevice == null)
            throw new Exception("Module not yet Loaded / Initialized. Use Load() first");

        //Change renderTarget resolution if different
        if (width != this.width || height != this.height)
        {
            UpdateResolution(width, height);

            //Adjust the blur
            glowRadiusMultiplier = (float)width / inputTexture.Width;

            //Update variables with the multiplier
            SetGlowPreset(GlowPreset);
        }

        graphicsDevice.RasterizerState = RasterizerState.CullNone;
        graphicsDevice.BlendState = BlendState.Opaque;


        //Extract the bright values which are above the Threshold and save them to 0
```

```
graphicsDevice.SetRenderTarget(glowRenderTarget2D0);

GlowScreenTexture = inputTexture;
GlowInverseResolution = new Vector2(1.0f / this.width, 1.0f / this.height);

if (glowUseLuminance) glowPassExtractLuminance.Apply();
else glowPassExtract.Apply();
screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);
```

Now add the if statements for the upsampling and downsampling to the draw function:

```
_____//Now downsample to the next lower texture
if (glowDownsamplePasses > 0)
{
   //DOWNSAMPLE TO 1
   graphicsDevice.SetRenderTarget(glowRenderTarget2D1);

   GlowScreenTexture = glowRenderTarget2D0;
   //Pass
   glowPassDownsample.Apply();
   screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);

   if (glowDownsamplePasses > 1)
   {
      GlowInverseResolution *= 2;

      //DOWNSAMPLE TO 2
      graphicsDevice.SetRenderTarget(glowRenderTarget2D2);

      GlowScreenTexture = glowRenderTarget2D1;
      //Pass
      glowPassDownsample.Apply();
      screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);

      if (glowDownsamplePasses > 2)
      {
         GlowInverseResolution *= 2;

         //DOWNSAMPLE TO 3
         graphicsDevice.SetRenderTarget(glowRenderTarget2D3);
```

```
                GlowScreenTexture = glowRenderTarget2D2;
                //Pass
                glowPassDownsample.Apply();
                screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);

                if (glowDownsamplePasses > 3)
                {
                    GlowInverseResolution *= 2;

                    //DOWNSAMPLE TO 4
                    graphicsDevice.SetRenderTarget(glowRenderTarget2D4);

                    GlowScreenTexture = glowRenderTarget2D3;
                    //Pass
                    glowPassDownsample.Apply();
                    screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);

                    if (glowDownsamplePasses > 4)
                    {
                        GlowInverseResolution *= 2;

                        //DOWNSAMPLE TO 5
                        graphicsDevice.SetRenderTarget(glowRenderTarget2D5);

                        GlowScreenTexture = glowRenderTarget2D4;
                        //Pass
                        glowPassDownsample.Apply();
                        screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1,
Vector2.One);

                        ChangeBlendState();

                        //UPSAMPLE TO 4
                        graphicsDevice.SetRenderTarget(glowRenderTarget2D4);
                        GlowScreenTexture = glowRenderTarget2D5;

                        GlowStrength = glowStrength5;
                        GlowRadius = glowRadius5;
                        if (glowUseLuminance) glowPassUpsampleLuminance.Apply();
```

```
                    else glowPassUpsample.Apply();
                    screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1,
Vector2.One);

                    GlowInverseResolution /= 2;
                }

                ChangeBlendState();

                //UPSAMPLE TO 3
                graphicsDevice.SetRenderTarget(glowRenderTarget2D3);
                GlowScreenTexture = glowRenderTarget2D4;

                GlowStrength = glowStrength4;
                GlowRadius = glowRadius4;
                if (glowUseLuminance) glowPassUpsampleLuminance.Apply();
                else glowPassUpsample.Apply();
                screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);

                GlowInverseResolution /= 2;

            }

            ChangeBlendState();

            //UPSAMPLE TO 2
            graphicsDevice.SetRenderTarget(glowRenderTarget2D2);
            GlowScreenTexture = glowRenderTarget2D3;

            GlowStrength = glowStrength3;
            GlowRadius = glowRadius3;
            if (glowUseLuminance) glowPassUpsampleLuminance.Apply();
            else glowPassUpsample.Apply();
            screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);

            GlowInverseResolution /= 2;

        }

        ChangeBlendState();
```

```
            //UPSAMPLE TO 1
            graphicsDevice.SetRenderTarget(glowRenderTarget2D1);
            GlowScreenTexture = glowRenderTarget2D2;

            GlowStrength = glowStrength2;
            GlowRadius = glowRadius2;
            if (glowUseLuminance) glowPassUpsampleLuminance.Apply();
            else glowPassUpsample.Apply();
            screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);

            GlowInverseResolution /= 2;
        }

        ChangeBlendState();

        //UPSAMPLE TO 0
        graphicsDevice.SetRenderTarget(glowRenderTarget2D0);
        GlowScreenTexture = glowRenderTarget2D1;

        GlowStrength = glowStrength1;
        GlowRadius = glowRadius1;

        if (glowUseLuminance) glowPassUpsampleLuminance.Apply();
        else glowPassUpsample.Apply();
        screenRenderer.RenderQuad(graphicsDevice, Vector2.One * -1, Vector2.One);
    }

    return glowRenderTarget2D0;
}

private void ChangeBlendState()
{
    graphicsDevice.BlendState = BlendState.AlphaBlend;
}
```

Add function to change Resolution to the screen:

```
    public void UpdateResolution(int width, int height)
{
    this.width = width;
```

```
        this.height = height;

        if (glowRenderTarget2D0 != null)
        {
            Dispose();
        }

        glowRenderTarget2D0 = new RenderTarget2D(graphicsDevice,
            (int)(width),
            (int)(height), false, renderTargetFormat, DepthFormat.None, 0,
RenderTargetUsage.DiscardContents);
        glowRenderTarget2D1 = new RenderTarget2D(graphicsDevice,
            (int)(width / 2),
            (int)(height / 2), false, renderTargetFormat, DepthFormat.None, 0,
RenderTargetUsage.PreserveContents);
        glowRenderTarget2D2 = new RenderTarget2D(graphicsDevice,
            (int)(width / 4),
            (int)(height / 4), false, renderTargetFormat, DepthFormat.None, 0,
RenderTargetUsage.PreserveContents);
        glowRenderTarget2D3 = new RenderTarget2D(graphicsDevice,
            (int)(width / 8),
            (int)(height / 8), false, renderTargetFormat, DepthFormat.None, 0,
RenderTargetUsage.PreserveContents);
        glowRenderTarget2D4 = new RenderTarget2D(graphicsDevice,
            (int)(width / 16),
            (int)(height / 16), false, renderTargetFormat, DepthFormat.None, 0,
RenderTargetUsage.PreserveContents);
        glowRenderTarget2D5 = new RenderTarget2D(graphicsDevice,
            (int)(width / 32),
            (int)(height / 32), false, renderTargetFormat, DepthFormat.None, 0,
RenderTargetUsage.PreserveContents);
    }
```

Dispose of the glow render targets because the trash collector doesn't do it automatically:

```
_____///Dispose our RenderTargets
    public void Dispose()
    {
        glowRenderTarget2D0.Dispose();
        glowRenderTarget2D1.Dispose();
```

```
    glowRenderTarget2D2.Dispose();
    glowRenderTarget2D3.Dispose();
    glowRenderTarget2D4.Dispose();
    glowRenderTarget2D5.Dispose();
}
```

D. ScreenRenderer.cs

Add these using statements:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
```

Class to set up screen space:

```
public class ScreenRenderer
    {
        //buffers for rendering the quad
        private readonly VertexPositionTexture[] vertexBuffer;
        private readonly short[] indexBuffer;


        public ScreenRenderer(GraphicsDevice graphicsDevice)
        {
            vertexBuffer = new VertexPositionTexture[4];
            vertexBuffer[0] = new VertexPositionTexture(new Vector3(-1, 1, 1), new
Vector2(0, 0));
            vertexBuffer[1] = new VertexPositionTexture(new Vector3(1, 1, 1), new
Vector2(1, 0));
            vertexBuffer[2] = new VertexPositionTexture(new Vector3(-1, -1, 1), new
Vector2(0, 1));
            vertexBuffer[3] = new VertexPositionTexture(new Vector3(1, -1, 1), new
Vector2(1, 1));

            indexBuffer = new short[] { 0, 3, 2, 0, 1, 3 };
        }

        public void RenderQuad(GraphicsDevice graphicsDevice, Vector2 v1, Vector2 v2)
        {
            vertexBuffer[0].Position.X = v1.X;
            vertexBuffer[0].Position.Y = v2.Y;

            vertexBuffer[1].Position.X = v2.X;
```

```
            vertexBuffer[1].Position.Y = v2.Y;

            vertexBuffer[2].Position.X = v1.X;
            vertexBuffer[2].Position.Y = v1.Y;

            vertexBuffer[3].Position.X = v2.X;
            vertexBuffer[3].Position.Y = v1.Y;

            graphicsDevice.DrawUserIndexedPrimitives
               (PrimitiveType.TriangleList, vertexBuffer, 0, 4, indexBuffer, 0, 2);
        }
    }
```

E. Main Program (Final.cs)

Add these using statements:

```
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
```

Variables:

```
    private int width = 1920;
    private int height = 1080;

    private float avgFps = 0;

    private bool isActiveWindow = true;

    private Texture2D ufo;
    private Texture2D sunset;
    private Texture2D sampleImage;
    private Texture2D currentImage;

    private SpriteFont spriteFont;
    private StringBuilder info;

    private GlowFilter glowFilter;
```

Initialize:

```
        Graphics.GraphicsProfile = GraphicsProfile.HiDef;
```

```
        Graphics.PreferredBackBufferWidth = width;
        Graphics.PreferredBackBufferHeight = height;

        IsFixedTimeStep = false;
        Graphics.SynchronizeWithVerticalRetrace = true;

        IsMouseVisible = true;

        //Active window
        Activated += IsActivated;
        Deactivated += IsDeactivated;
```

Load the content of the controls you make later and of the pictures you want to pass through the effect:

```
        ufo = Content.Load<Texture2D>("???");
        sampleImage = Content.Load<Texture2D>("???");
        sunset = Content.Load<Texture2D>("???");
        spriteFont = Content.Load<SpriteFont>("Font");
        info = new StringBuilder("Use F1 - F5 for different glow presets." + "\n" +
"???");

        //Load Glow filter
        glowFilter = new GlowFilter();
        glowFilter.Load(GraphicsDevice, Content, width, height);

        glowFilter.GlowPreset = GlowFilter.GlowPresets.SuperWide;

        currentImage = ufo;
```

Unload the dispose function built in the filter:

```
protected override void UnloadContent()
    {
        glowFilter.Dispose();
    }
```

Create controls to control the strength, threshold, and current image here is the update to change preset:

```
if (!isActiveWindow) return;
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
```

```
            Exit();
        KeyboardState state = Keyboard.GetState();
        MouseState mstate = Mouse.GetState();

        if (state.IsKeyDown(Keys.F1)) glowFilter.GlowPreset =
GlowFilter.GlowPresets.Wide;
        if (state.IsKeyDown(Keys.F2)) glowFilter.GlowPreset =
GlowFilter.GlowPresets.SuperWide;
        if (state.IsKeyDown(Keys.F3)) glowFilter.GlowPreset =
GlowFilter.GlowPresets.Focussed;
        if (state.IsKeyDown(Keys.F4)) glowFilter.GlowPreset =
GlowFilter.GlowPresets.Small;
        if (state.IsKeyDown(Keys.F5)) glowFilter.GlowPreset =
GlowFilter.GlowPresets.Cheap;
```

The fps might give you trouble if you add too much glow effect or if your hardware is not up to date so add this to track the fps and other properties on the window title bar:

```
float fps = 0;
        if (gameTime.ElapsedGameTime.TotalMilliseconds > 0)
            fps = (float)Math.Round(1000 /
(gameTime.ElapsedGameTime.TotalMilliseconds), 1);

        if (avgFps < 0.01f) avgFps = fps;

        avgFps = avgFps * 0.95f + fps * 0.05f;

        Window.Title = "GlowFilter Preset: " + glowFilter.GlowPreset +
                    " with " + glowFilter.glowDownsamplePasses +
                    " Passes | Threshold: " + Math.Round(glowFilter.GlowThreshold, 2) +
                    "Strength: " + Math.Round(glowFilter.glowStrengthMultiplier, 2) +
                    " | Streaks: " + glowFilter.GlowStreakLength +
                    " | FPS : " + avgFps;
```

Main Draw function to show everything added together:

```
protected override void Draw(GameTime gameTime)
    {
        if (!isActiveWindow) return;
        GraphicsDevice.Clear(Color.CornflowerBlue);
```

```
            int w = width;
            int h = height;

            Texture2D bloom = glowFilter.Draw(currentImage, w, h);

            GraphicsDevice.SetRenderTarget(null);

            spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.Additive);

            spriteBatch.Draw(currentImage, new Rectangle(0, 0, width, height),
Color.White);
            spriteBatch.Draw(bloom, new Rectangle(0, 0, width, height), Color.White);

            spriteBatch.DrawString(spriteFont, info, Vector2.One, Color.White);

            spriteBatch.End();
            base.Draw(gameTime);
        }
```

Finally add a check to make sure the window is currently active to be able to change the effect:

```
        private void IsDeactivated(object sender, EventArgs e)
        {
            isActiveWindow = false;
        }

        private void IsActivated(object sender, EventArgs e)
        {
            isActiveWindow = true;
        }
```

F. Main Exercise

Go back to the highlighted sections to make the other presets for the glow effect. Also add controls to the main program to change the image in real time as well as change the image itself. In the load function and with some changes to the variables add your own images to see how the effect grabs the brightest point in any photo and can apply any of the glow presets to it. The streak length of the glow should be able to be changed up and down and the threshold/strength of the glow should be controlled by the mouse.

Example Outputs:

*** IMPORTANT ***

Complete the exercise, and submit a zipped file including the solution (.sln) file and the project folders to course online. The submission item is located in the "Quiz and Lab" section. Each lab has 10 points. If you complete the exercise in class time, the full points will be assigned. The late submission is accepted just before the next class with 2 points reductions, because the solution is demonstrated in the next class.