Multi Pass Effects in Candera > Examples for Global 3D Effect Shaders

Description

This chapter describes how global multi pass effects in Candera can be achieved and special 3D effect shaders techniques supported by Candera.

Depth of Field

Description

This chapter describes how to generate the **Depth of Field** effect using the global multipass appearance technique supported by Candera.

Introduction

Depth of Field

The Depth-of-Field effect simulates the natural blurring of foreground and background scene elements when viewed through a camera lens. The effect firstly separates the scene in Z order and then blurs the foreground and background images according to the values set in the Depth of Field effect parameters. The final image is composited from the processed originals.

In Candera, the effect is realized using a global multipass technique which implies the rendering in four steps: [1]

- Render pass 1 - rendering the sharp scene, storing depth values in alpha channel
- Render pass 2 - blurring the result of the first render pass horizontally
- Render pass 3 - blurring the result of the second render pass vertically
- Render pass 4 - combine sharp and blurred picture, using focus-, focus range- and stored depth values, in order to realize the final result

Workflow

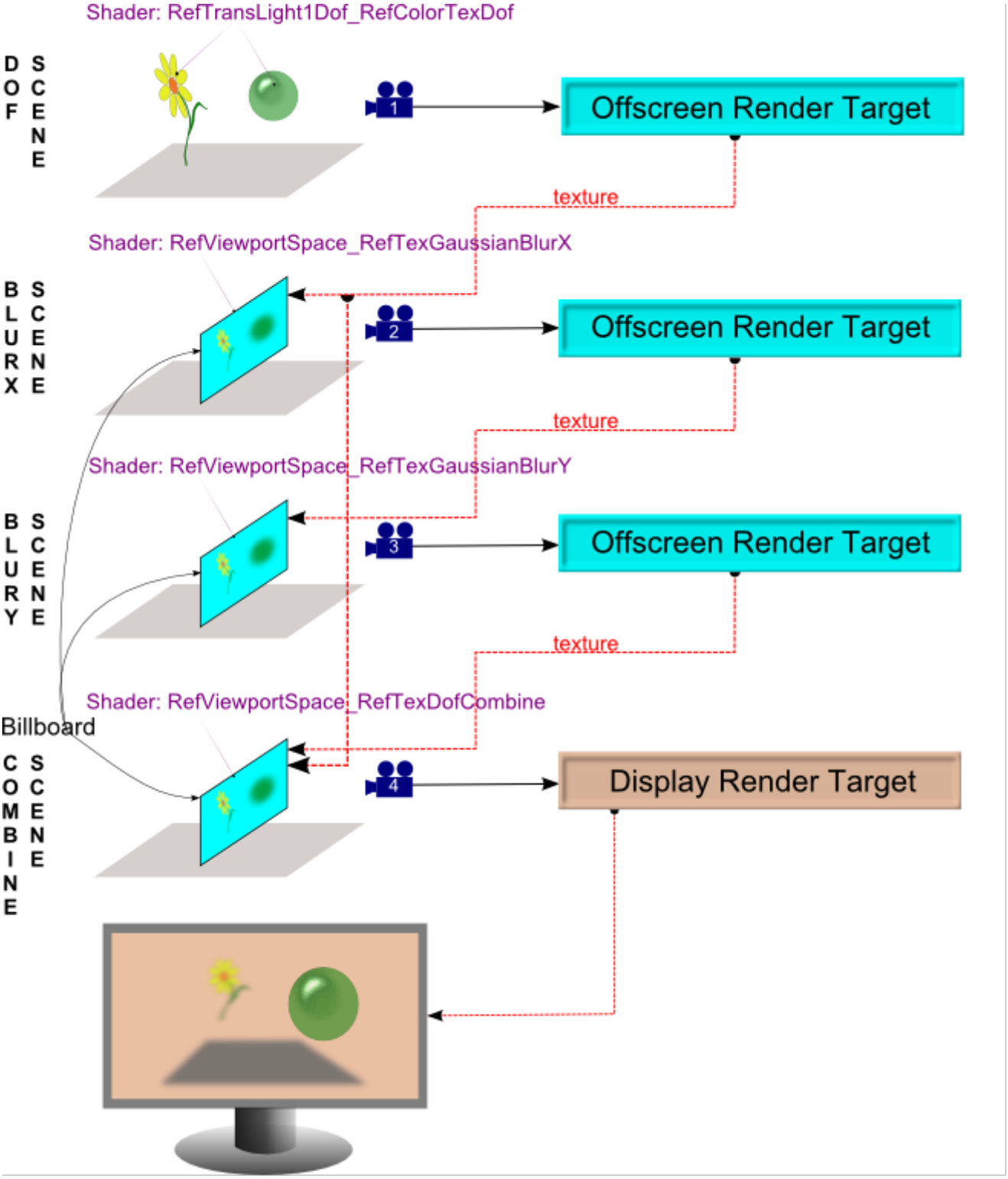
Scenes

For an easier understanding on how to realize the Depth of Field effect we will use four scenes, each scene will correspond to a rendering step.

First of all, the original main scene is taken and used as a texture on a billboard. The depth values of the images are saved in the alpha value in the offscreen frame buffer. In the second and third steps, the blurring is being done. For reasons of the performance, this step is parted into two steps: The resulting billboard of the first workflow step is blurred horizontally using a fragment shader. The resulting image is used as texture input for the next scene, which is subsequently blurred vertically.
In the final step, the blurred images are combined with the sharp images from the first offscreen render target. The result is rendered as texture on a billboard placed in the final scene. A fourth camera displays the resulting Depth of Field effect applied on the main scene.

> The Billboards should be rendered such that they cover the whole screen

The technique is also described schematically in the diagram below: [1]



> Remember to set a Candera::RenderOrder object for each scene.

The sections below offer you some extra information needed to correctly set up the effect.

Offscreen Render Targets

Create and configure all offscreen render targets that shall be used for cameras 1 - 3 as shown below:

```
    m_offscreenRenderTargetDof = Base::CreateGraphicDeviceUnit(DevicePackageDescriptor::Offscree
    if (m_offscreenRenderTargetDof == 0) {
        return false;
    }

    sprintf(m_rtProperties.m_width, "%d", m_gdu->ToRenderTarget3D()->GetWidth());
    sprintf(m_rtProperties.m_height, "%d", m_gdu->ToRenderTarget3D()->GetHeight());
    sprintf(m_rtProperties.m_colorFormat, "%d", RgbaUnpackedColorFormat);
    sprintf(m_rtProperties.m_depthFormat, "%d", Depth32Format);

    const MetaInfo::GraphicDeviceUnitMetaInfo *meta = DevicePackageDescriptor::GetMetaInformatic
    meta->LookupItem("Width") && meta->LookupItem("Width")->Set(m_offscreenRenderTargetDof, m_rt
    meta->LookupItem("Height") && meta->LookupItem("Height")->Set(m_offscreenRenderTargetDof, m_
    meta->LookupItem("ColorFormat") && meta->LookupItem("ColorFormat")->Set(m_offscreenRenderTar
    meta->LookupItem("DepthFormat") && meta->LookupItem("DepthFormat")->Set(m_offscreenRenderTar

#if CANDERA_DEVICE_IMX6
    static_cast<iMX6FrameBufferObject*>(m_offscreenRenderTargetDof)->GetProperties().SetOwner(m_
#endif
    m_offscreenRenderTargetDof->Upload();
```

Shader Parameter Setter

The meshes from each scene will use a dedicated shader which will set specific uniforms for each scene. The uniforms needed to be set are:

- Scaled depth value (u_depthScale): factor used to scale the depth value
- Circle of confusion (u_CocScale): factor used to scale the size of circle of confusion
- Focus range (u_Range): factor used in the computation of the weight of sharpness / blurriness
- Focus (u_Focus): factor used in the computation of the weight of sharpness / blurriness

```
    // DoF Scene
    m_objectSps = GenericShaderParamSetter::Create();
    m_objectSps->SetLightsCoordinateSpace(Light::Object);
    m_objectSps->SetModelMatrix4Enabled(false);
    m_objectSps->SetNormalModelMatrix3Enabled(false);
    m_objectSps->SetUniform("u_depthScale",Shader::Float, &m_depthScale);

    // Blur Scene vertically
    m_blurXSps = GenericShaderParamSetter::Create();
    m_blurXSps->SetModelViewProjectionMatrix4Enabled(false);
    m_blurXSps->SetUniform("u_blurFactor",Shader::Float, &m_blurFactor);
    m_blurXSps->SetUniform("u_offsetLeft", Shader::Float, &m_viewportLeft);
    m_blurXSps->SetUniform("u_offsetTop", Shader::Float, &m_viewportTop);
```

```
    // Blur Scene horizontally
    m_blurYSps = GenericShaderParamSetter::Create();
    m_blurYSps->SetModelViewProjectionMatrix4Enabled(false);
    m_blurYSps->SetUniform("u_blurFactor",Shader::Float, &m_blurFactor);
    m_blurYSps->SetUniform("u_offsetLeft", Shader::Float, &m_viewportLeft);
    m_blurYSps->SetUniform("u_offsetTop", Shader::Float, &m_viewportTop);

    // Combine Scene
    m_dofCombineSps = GenericShaderParamSetter::Create();
    m_dofCombineSps->SetModelViewProjectionMatrix4Enabled(false);
    m_dofCombineSps->SetUniform("u_Range",Shader::Float, &m_range);
    m_dofCombineSps->SetUniform("u_Focus",Shader::Float, &m_focus);
    m_dofCombineSps->SetUniform("u_offsetLeft", Shader::Float, &m_viewportLeft);
    m_dofCombineSps->SetUniform("u_offsetTop", Shader::Float, &m_viewportTop);
```

Camera

The cameras from the all three scenes should use perspective projection.

```
    SharedPointer<PerspectiveProjection> perspectiveProjection = PerspectiveProjection::Create
();
    perspectiveProjection->SetNearZ(0.001F);
    perspectiveProjection->SetFarZ(100.0F);
    perspectiveProjection->SetFovYDegrees(45.0F);
    perspectiveProjection->SetAspectRatio(static_cast<Float>(dispSettings->width) / static_cast<
```

Set the clear mode for the first two cameras as below:

```
    m_clearMode.SetClearColor(Color(0.8f,0.8f,0.8f,1.0f));
    m_clearMode.SetColorClearEnabled(true);
    m_clearMode.SetDepthClearEnabled(true);
    m_clearMode.SetClearDepth(1.0f);
```

Billboard Textures

This code snippet shows you how to create the textures for the billboards. First create a Candera::ProxyTextureImage:

```
    SharedPointer<ProxyTextureImage> proxyTexImgBlurredX = ProxyTextureImage::Create
(m_offscreenRenderTargetBlurX->ToImageSource3D());
    SharedPointer<ProxyTextureImage> proxyTexImgBlurredY = ProxyTextureImage::Create
(m_offscreenRenderTargetBlurY->ToImageSource3D());
```

Then create and configure the texture:

```
    SharedPointer<Texture> proxyTexBlurredX = Texture::Create();
    proxyTexBlurredX->SetTextureImage(proxyTexImgBlurredX);
    proxyTexBlurredX->SetMipMapFilter(Texture::MipMapNone);
```

```
proxyTexBlurredX->SetMagnificationFilter(Texture::MinMagNearest);
proxyTexBlurredX->SetMinificationFilter(Texture::MinMagNearest);
proxyTexBlurredX->SetWrapModeU(Texture::ClampToEdge);
proxyTexBlurredX->SetWrapModeV(Texture::ClampToEdge);

SharedPointer<Texture> proxyTexBlurredY = Texture::Create();
proxyTexBlurredY->SetTextureImage(proxyTexImgBlurredY);
proxyTexBlurredY->SetMipMapFilter(Texture::MipMapNone);
proxyTexBlurredY->SetMagnificationFilter(Texture::MinMagNearest);
proxyTexBlurredY->SetMinificationFilter(Texture::MinMagNearest);
proxyTexBlurredY->SetWrapModeU(Texture::ClampToEdge);
proxyTexBlurredY->SetWrapModeV(Texture::ClampToEdge);
```

Repeat these two steps for all textures.

> The billboard from the Combine Scene uses two textures. When you set the textures be sure that the *texture unit* value is set to "0" for the texture corresponding to the camera 1. Analogous, the other texture will have the *texture unit* value set to "1".

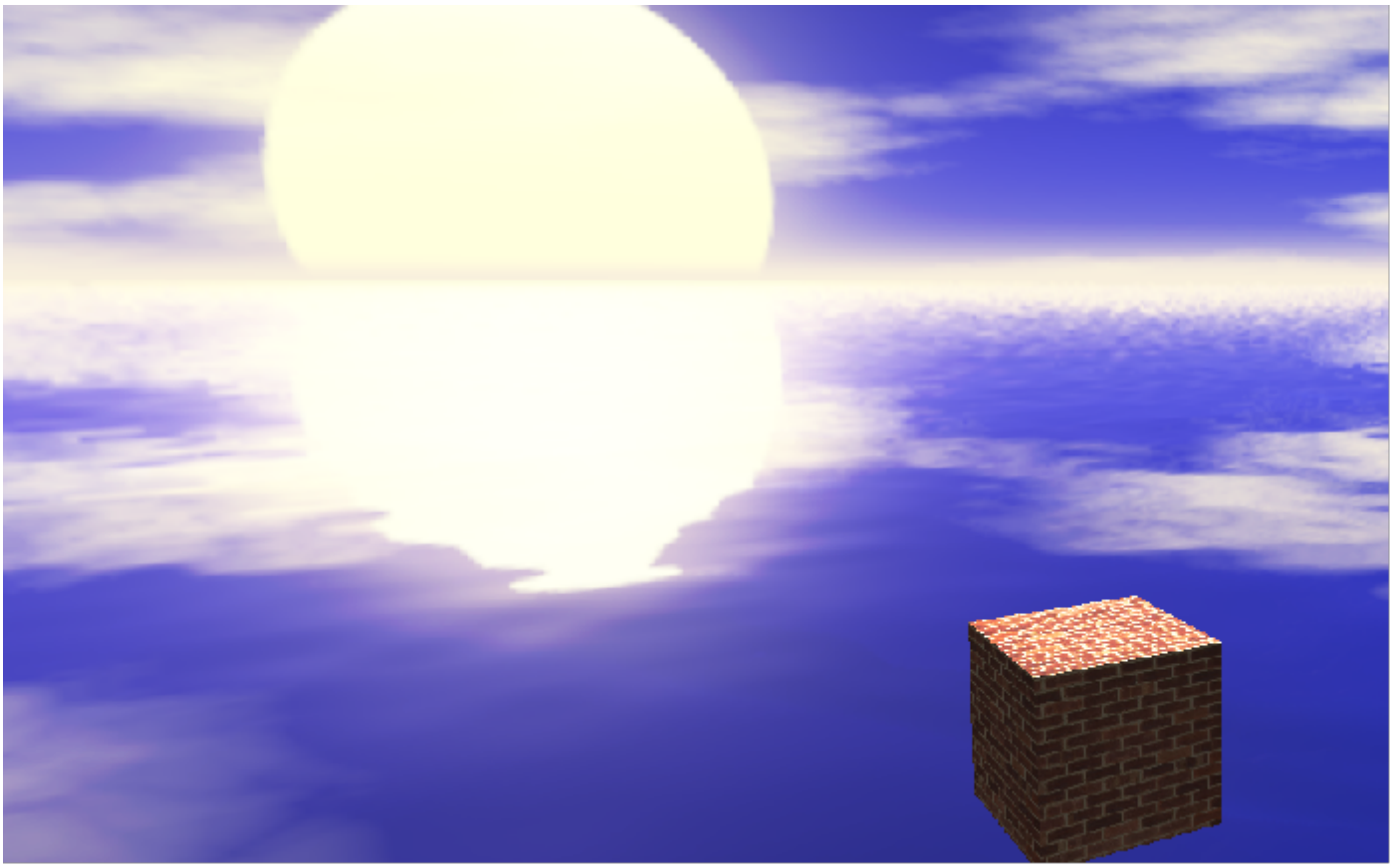
Placing Billboards in Scenes

See Shader Usage in Candera: Global Multi Pass Effects

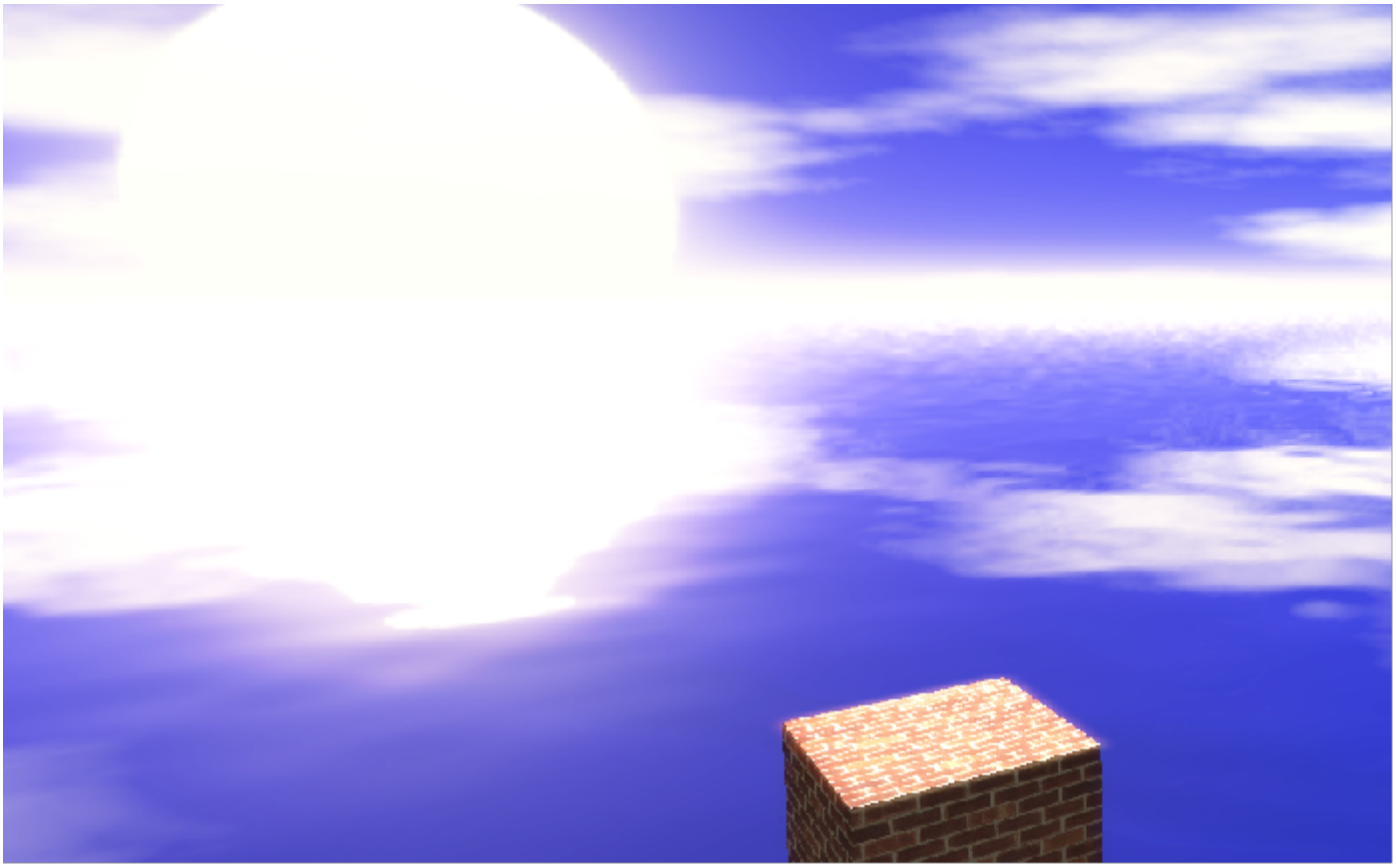
Example

Here is exemplified the **Bloom** effect on a scene.

First image shows the scene without the effect:



Second image shows same scene but with the effect:



References

[1] http://www.geeks3d.com/20090710/javascript-depth-of-field-effect

---

Bloom

Description

This chapter describes how to generate the **Bloom** effect using the global multipass appearance technique supported by Candera.

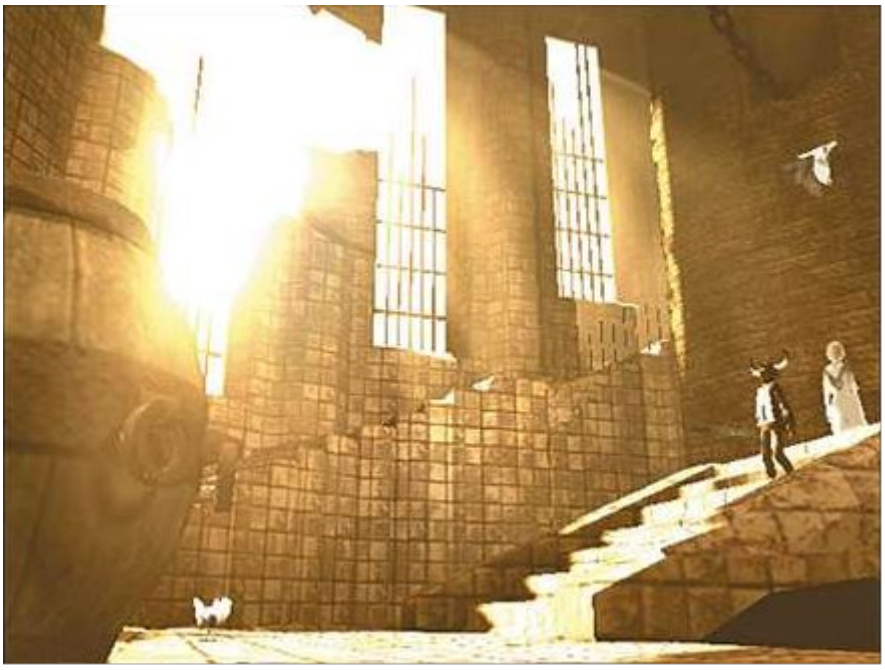
Introduction

Bloom

The Bloom-Effect is a shader effect used to simulate the appearance of very bright light. The effect firstly extracts the bright area of an image using a threshold filter. The extracted regions are further blurred in order to realize the simulation of light bleeding over darker regions.

The bloom effect is realized using a global multipass technique which implies the rendering in five steps: [1]

- Render pass 1 - rendering the sharp scene
- Render pass 2 - extract bright area using a threshold filter
- Render pass 3 - blurring the result of the second render pass horizontally
- Render pass 4 - blurring the result of the third render pass vertically
- Render pass 5 - combine sharp and blurred picture, using intensity as well as saturation of bloom texture and original scene texture, in order to generate the final result



Workflow

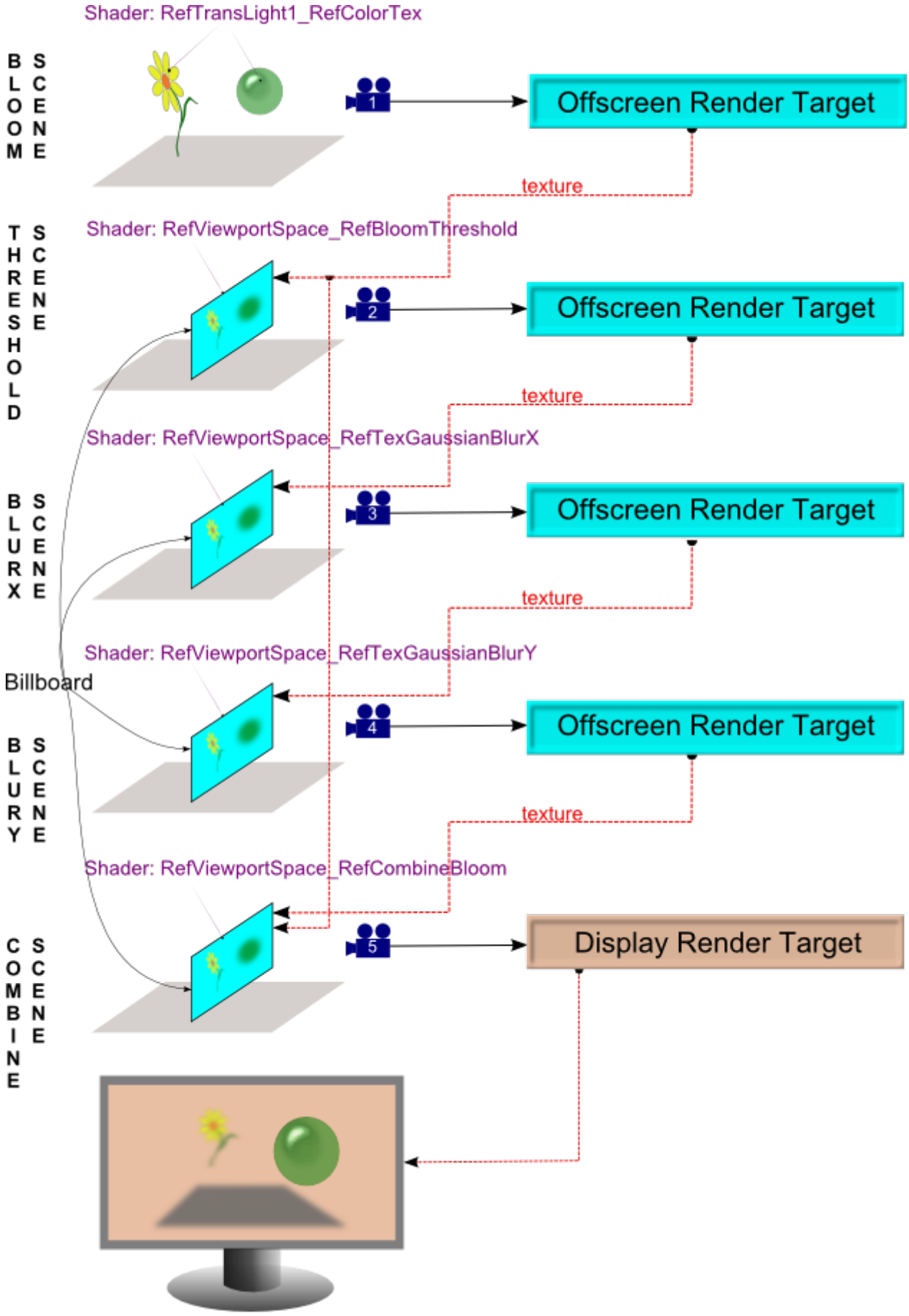
Scenes

For an easier understanding on how to realize the Bloom effect we will use five scenes, each scene will correspond to a rendering step.

First of all, the original main scene is taken and used as a texture on a billboard. This billboard is rendered with a special shader that will extract the bright area of the texture using a threshold filter.
In the second and third steps, the blurring is being done. For reasons of the performance, this step is parted into two steps: The resulting billboard of the first workflow step is rendered with a shader that will blur the images horizontally and these blurred images are rendered as texture on an another billboard placed in the third scene. The billboard of the third scene is then rendered with a shader that will blur the images vertically. This billboard is part of the fourth scene, viewed by a camera which is connected to a fourth offscreen render target.
In the final step, the images of the blurred, bright area of the scene are combined with the sharp image from the first offscreen render target. The result is rendered as texture on a billboard placed in the final fifth scene. A fifth camera displays the resulting Bloom effect applied on the main scene.

> The Billboards should be rendered such that they cover the whole screen

The technique is also described schematically in the diagram below:


Shader: RefTransLight1_RefColorTex
BLOOM SCENE
1
Offscreen Render Target
texture
Shader: RefViewportSpace_RefBloomThreshold
THRESHOLD SCENE
2
Offscreen Render Target
texture
Shader: RefViewportSpace_RefTexGaussianBlurX
BLURX SCENE
3
Offscreen Render Target
texture
Shader: RefViewportSpace_RefTexGaussianBlurY
Billboard
BLURY SCENE
4
Offscreen Render Target
texture
Shader: RefViewportSpace_RefCombineBloom
COMBINE SCENE
5
Display Render Target

> Remember to set a Candera::RenderOrder object for each scene.

The sections below offer you some extra information needed to correctly set up the effect.

Shader Parameter Setter

The meshes from each scene will use a dedicated shader which will set specific uniforms for each scene. The uniforms needed to be set are:

- Bloom Brightness Threshold value (u_brightnessThreshold): brightness threshold value to render the bloom shape. lower = darker parts of the image
- Blur Factor (u_blurFactor): factor used in the computation of the weight of sharpness / blurriness
- Intensity amount on original scene (u_originalIntensity): factor used to set the intensity of the original scene texture
- Saturation amount on original scene (u_originalSaturation): factor used to set the saturation of the original scene texture
- Bloom Intensity (u_bloomIntensity): bloom intensity factor used in the combining scene
- Bloom Saturation (u_bloomSaturation): bloom saturation value used in the combining scene

```
// Sharp Scene
m_objectSps = GenericShaderParamSetter::Create();
m_objectSps->SetLightsCoordinateSpace(Light::Object);
m_objectSps->SetModelMatrix4Enabled(false);
m_objectSps->SetNormalModelMatrix3Enabled(false);

// Bloom Threshold Scene
m_bloomThresholdSps = GenericShaderParamSetter::Create();
m_bloomThresholdSps->SetModelViewProjectionMatrix4Enabled(false);
m_bloomThresholdSps->SetUniform("u_brightnessThreshold",Shader::Float, &m_bloomBrightnessThr
m_bloomThresholdSps->SetUniform("u_offsetLeft", Shader::Float, &m_offsetLeft);
m_bloomThresholdSps->SetUniform("u_offsetTop", Shader::Float, &m_offsetTop);

// Blur Scene vertically
m_bloomBlurXSps = GenericShaderParamSetter::Create();
m_bloomBlurXSps->SetModelViewProjectionMatrix4Enabled(false);
m_blurFactor = 1.0F / m_gdu->ToRenderTarget3D()->GetWidth();
m_bloomBlurXSps->SetUniform("u_blurFactor",Shader::Float, &m_blurFactor);
m_bloomBlurXSps->SetUniform("u_offsetLeft", Shader::Float, &m_offsetLeft);
m_bloomBlurXSps->SetUniform("u_offsetTop", Shader::Float, &m_offsetTop);

// Blur Scene horizontally
m_bloomBlurYSps = GenericShaderParamSetter::Create();
m_bloomBlurYSps->SetModelViewProjectionMatrix4Enabled(false);
```

```
    m_blurFactor = 1.0F / m_gdu->ToRenderTarget3D()->GetHeight();
    m_bloomBlurYSps->SetUniform("u_blurFactor",Shader::Float, &m_blurFactor);
    m_bloomBlurYSps->SetUniform("u_offsetLeft", Shader::Float, &m_offsetLeft);
    m_bloomBlurYSps->SetUniform("u_offsetTop", Shader::Float, &m_offsetTop);

    // Combine Scene
    m_bloomCombineSps = GenericShaderParamSetter::Create();
    m_bloomCombineSps->SetModelViewProjectionMatrix4Enabled(false);
    m_bloomCombineSps->SetUniform("u_originalIntensity",Shader::Float, &m_originalIntensity);
    m_bloomCombineSps->SetUniform("u_originalSaturation",Shader::Float, &m_originialSaturation);
    m_bloomCombineSps->SetUniform("u_bloomIntensity",Shader::Float, &m_bloomIntensity);
    m_bloomCombineSps->SetUniform("u_bloomSaturation",Shader::Float, &m_bloomSaturation);
    m_bloomCombineSps->SetUniform("u_offsetLeft",Shader::Float, &m_offsetLeft);
    m_bloomCombineSps->SetUniform("u_offsetTop",Shader::Float, &m_offsetTop);
```

Camera

The cameras from all the five scenes should use perspective projection.

```
    SharedPointer<PerspectiveProjection> perspectiveProjection = PerspectiveProjection::Create
();
    perspectiveProjection->SetNearZ(0.001F);
    perspectiveProjection->SetFarZ(100.0F);
    perspectiveProjection->SetFovYDegrees(45.0F);
    perspectiveProjection->SetAspectRatio(static_cast<Float>(dispSettings->width) / static_cast<
```

Set the clear mode for the cameras as below:

```
    m_clearMode.SetClearColor(Color(0.8f,0.8f,0.8f,1.0f));
    m_clearMode.SetColorClearEnabled(true);
    m_clearMode.SetDepthClearEnabled(true);
    m_clearMode.SetClearDepth(1.0f);
```

Billboard Textures

This code snippet shows you how to create the textures for the billboards. First create a Candera::ProxyTextureImage:

```
    SharedPointer<ProxyTextureImage> proxyTexImgBloomThreshold = ProxyTextureImage::Create
(m_offscreenRenderTargetThreshold->ToImageSource3D());
    SharedPointer<ProxyTextureImage> proxyTexImgBlurredX = ProxyTextureImage::Create
(m_offscreenRenderTargetBlurX->ToImageSource3D());
    SharedPointer<ProxyTextureImage> proxyTexImgBlurredY = ProxyTextureImage::Create
(m_offscreenRenderTargetBlurY->ToImageSource3D());
```

Then create and configure the texture:

```
SharedPointer<Texture> proxyTexBloomThreshold = Texture::Create();
proxyTexBloomThreshold->SetTextureImage(proxyTexImgBloomThreshold);
proxyTexBloomThreshold->SetMipMapFilter(Texture::MipMapNone);
proxyTexBloomThreshold->SetMagnificationFilter(Texture::MinMagNearest);
proxyTexBloomThreshold->SetMinificationFilter(Texture::MinMagNearest);
proxyTexBloomThreshold->SetWrapModeU(Texture::ClampToEdge);
proxyTexBloomThreshold->SetWrapModeV(Texture::ClampToEdge);

SharedPointer<Texture> proxyTexBlurredX = Texture::Create();
proxyTexBlurredX->SetTextureImage(proxyTexImgBlurredX);
proxyTexBlurredX->SetMipMapFilter(Texture::MipMapNone);
proxyTexBlurredX->SetMagnificationFilter(Texture::MinMagNearest);
proxyTexBlurredX->SetMinificationFilter(Texture::MinMagNearest);
proxyTexBlurredX->SetWrapModeU(Texture::ClampToEdge);
proxyTexBlurredX->SetWrapModeV(Texture::ClampToEdge);

SharedPointer<Texture> proxyTexBlurredY = Texture::Create();
proxyTexBlurredY->SetTextureImage(proxyTexImgBlurredY);
proxyTexBlurredY->SetMipMapFilter(Texture::MipMapNone);
proxyTexBlurredY->SetMagnificationFilter(Texture::MinMagNearest);
proxyTexBlurredY->SetMinificationFilter(Texture::MinMagNearest);
proxyTexBlurredY->SetWrapModeU(Texture::ClampToEdge);
proxyTexBlurredY->SetWrapModeV(Texture::ClampToEdge);
```

> The billboard from the Combine Scene uses two textures. When you set the textures be sure that the *texture unit* value is set to "0" for the texture corresponding to the camera 1. Analogous, the other texture will have the *texture unit* value set to "1".

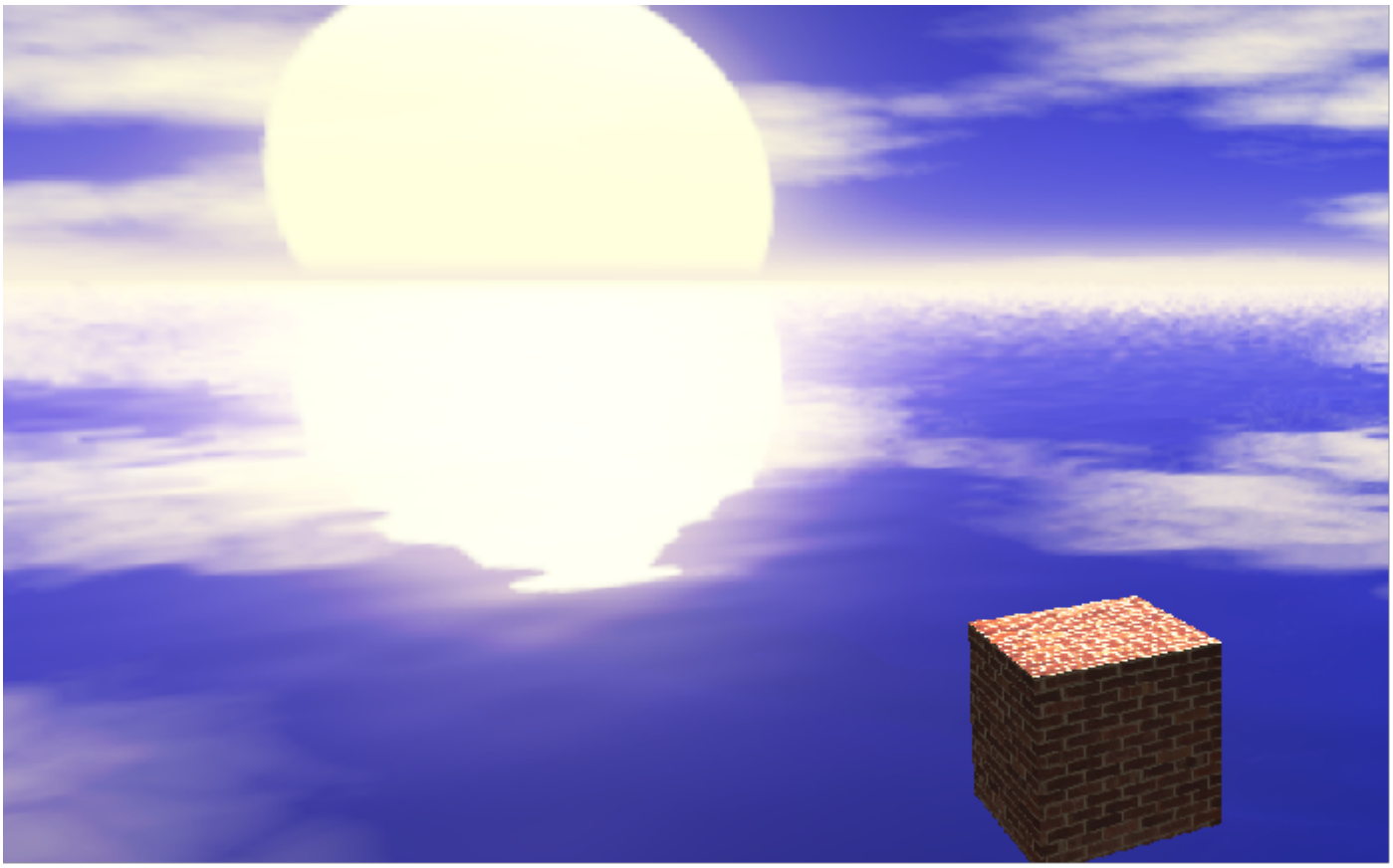
Placing Billboards in Scenes

See Shader Usage in Candera: Global Multi Pass Effects

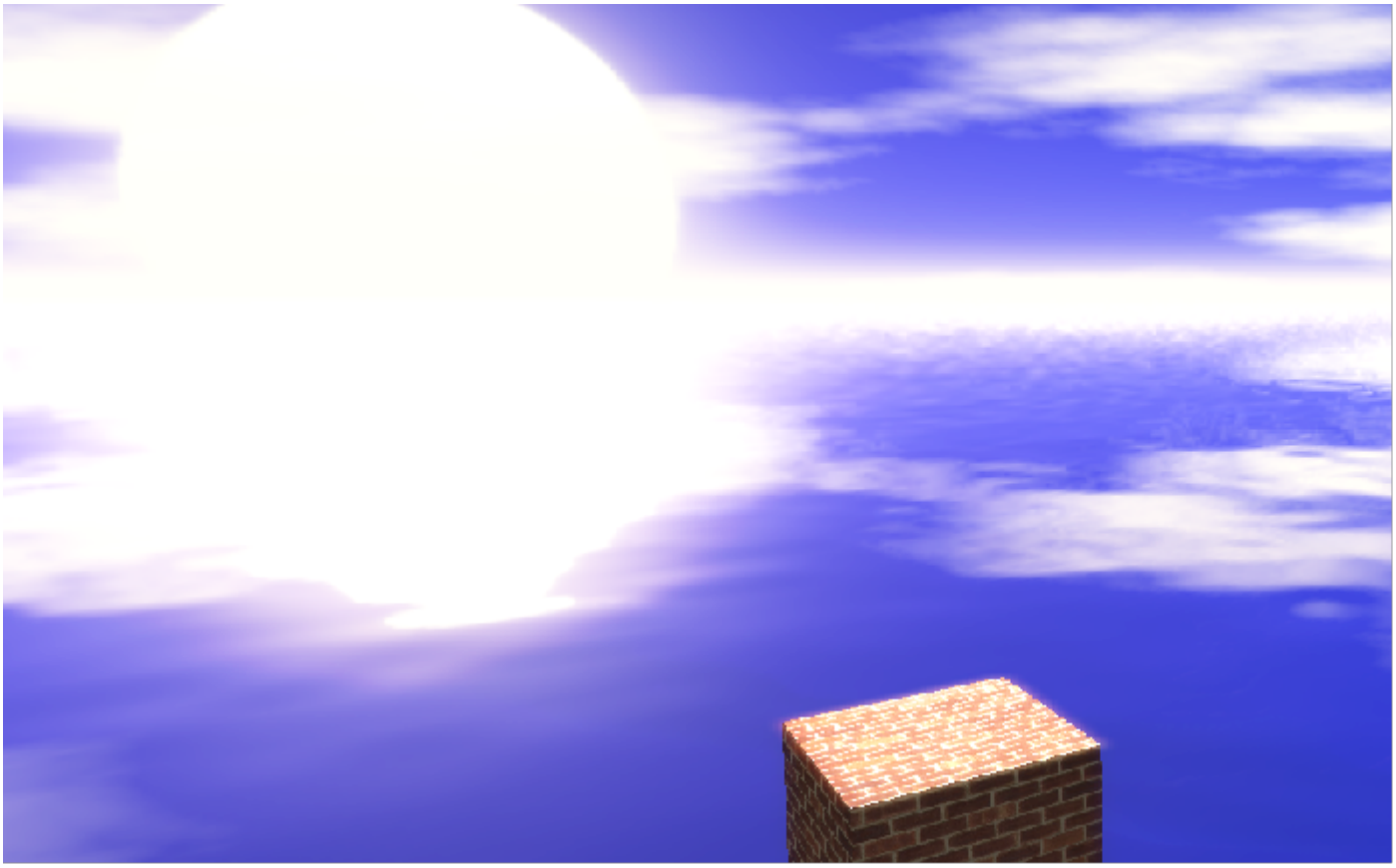
Example

Here is exemplified the **Bloom** effect on a scene.

First image shows the scene without the effect:



Second image shows same scene but with the effect:



Revision #12
Created 2023-02-22 08:39:10 UTC by Tsuyoshi.Kato
Updated 2025-01-10 14:13:53 UTC by Elisabeth Zauner