Real-time Graphics

7. Post-Processing

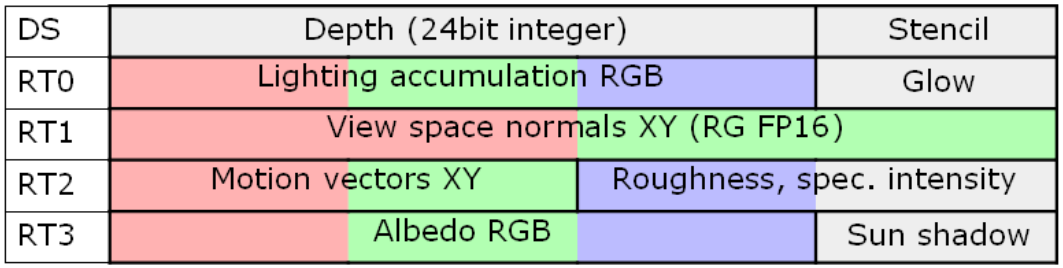
Martin Samuelčík

Post processing

Blizzard

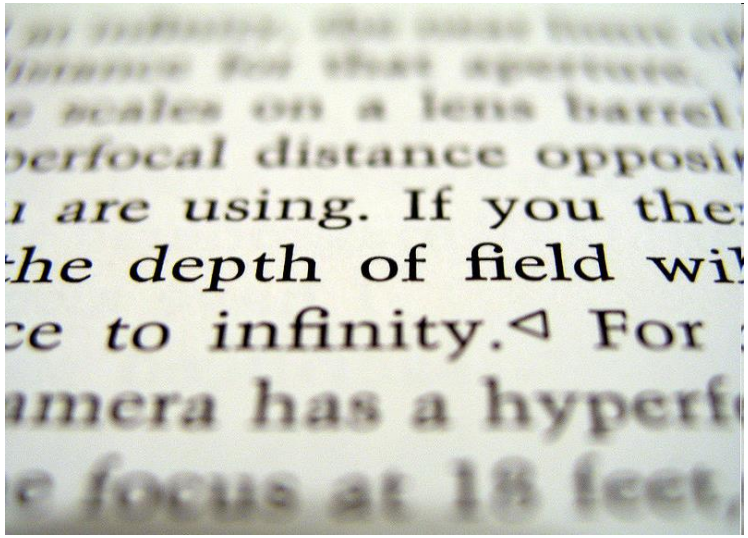
Post processing

- Additional visual effects after scene rendering
  - Depth of Field
  - Motion Blur
  - High Dynamic Range
  - Glow / Bloom / Glare
  - SSAO, edge filter, fog, …
- Mostly prepared and computed in screen space
- Several per-pixel information are needed
  - Color buffers
  - Depth buffer
  - Normal buffer
  - Motion vectors
- Deferred rendering



Depth of Field

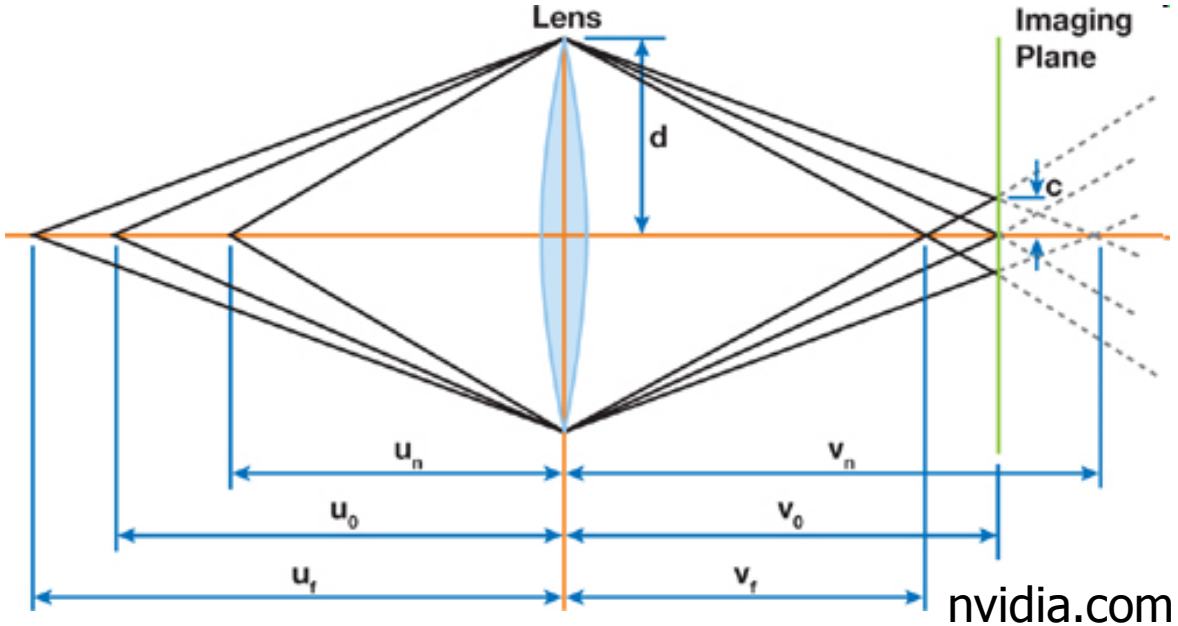
- Distance between the nearest and farthest objects in a scene that appear acceptably sharp in an image
- Emphasizing the object vs. sharp picture



wikipedia.org

Depth of Field

- u_0 is in focus
- u_f,u_n map to a circle of confusion(CoC) with radius *c*
- Depth of field = *c* is sufficiently small



nvidia.com

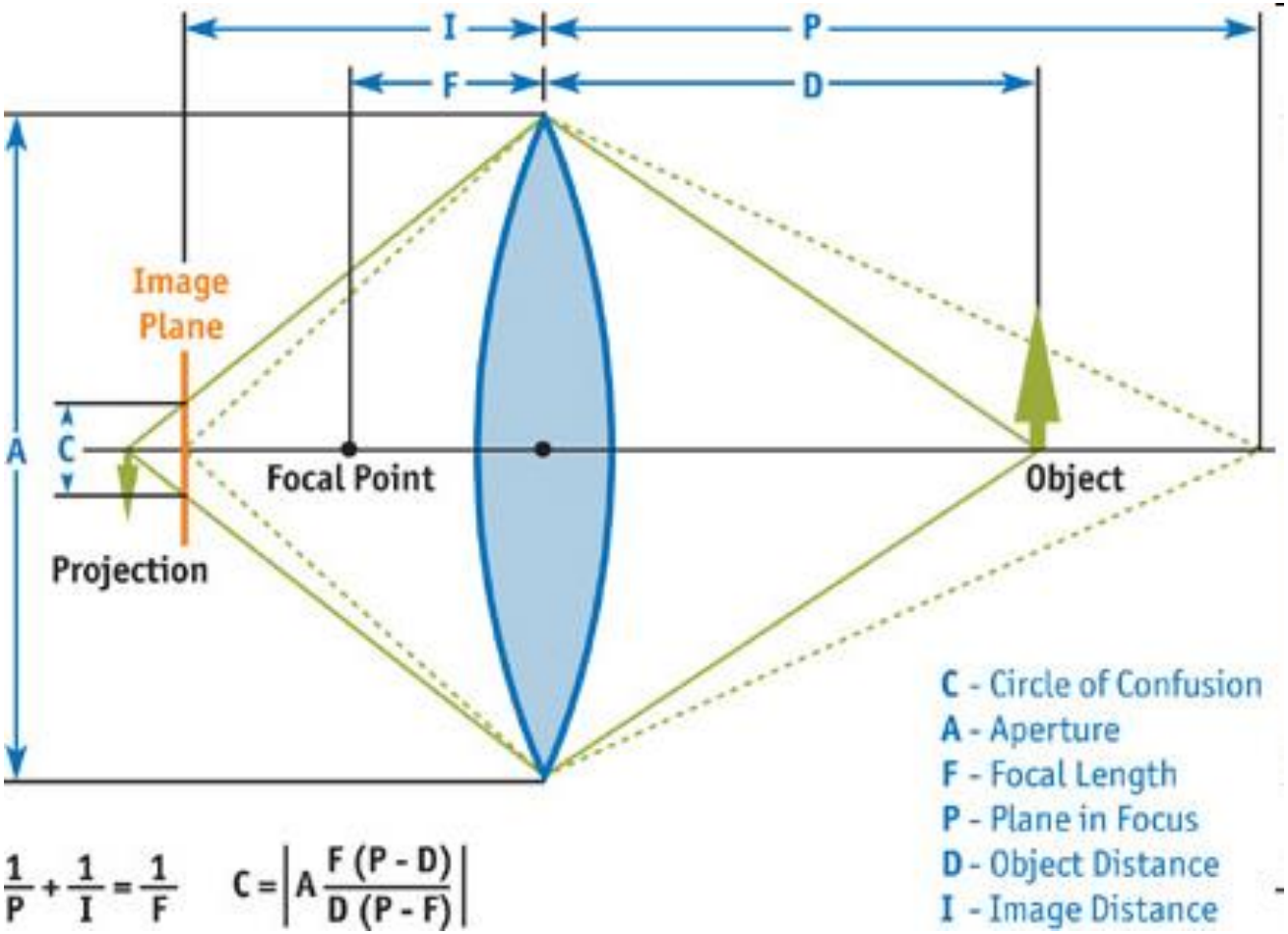
$$\frac{1}{u}+\frac{1}{v}=\frac{1}{f}$$

$$\frac{v_n - v_o}{v_n} = \frac{c}{d} = \frac{v_o - v_f}{v_f}$$

Circle of confusion for point *p*

$$c = d \times \left| \frac{v_p - v_o}{v_p} \right|$$

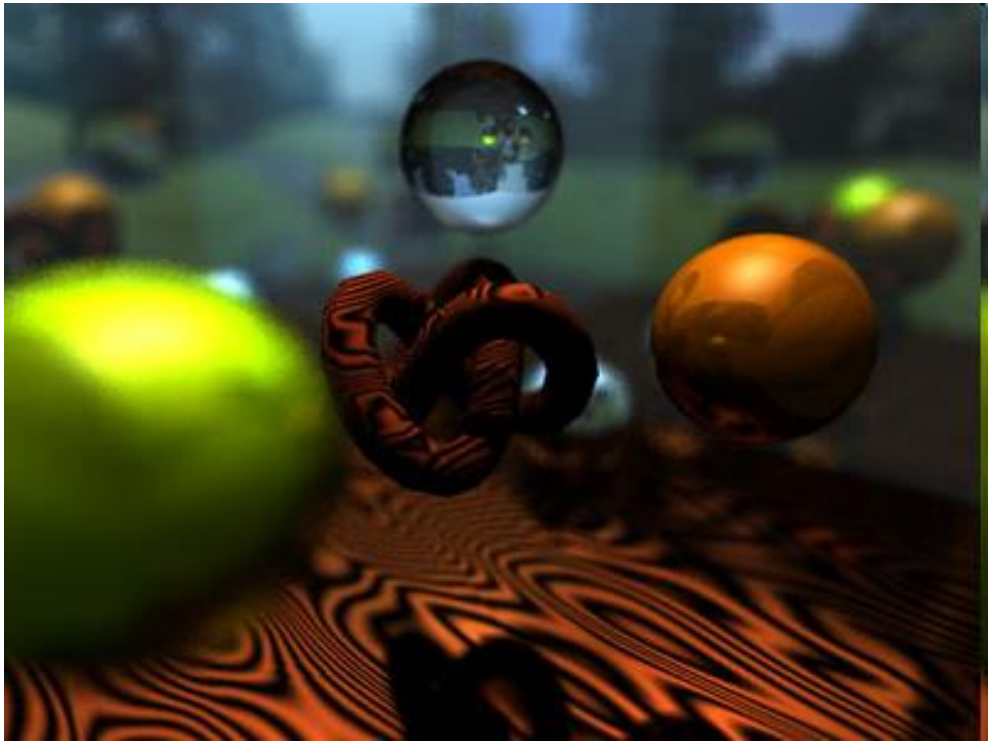
Depth of Field



.nvidia.com

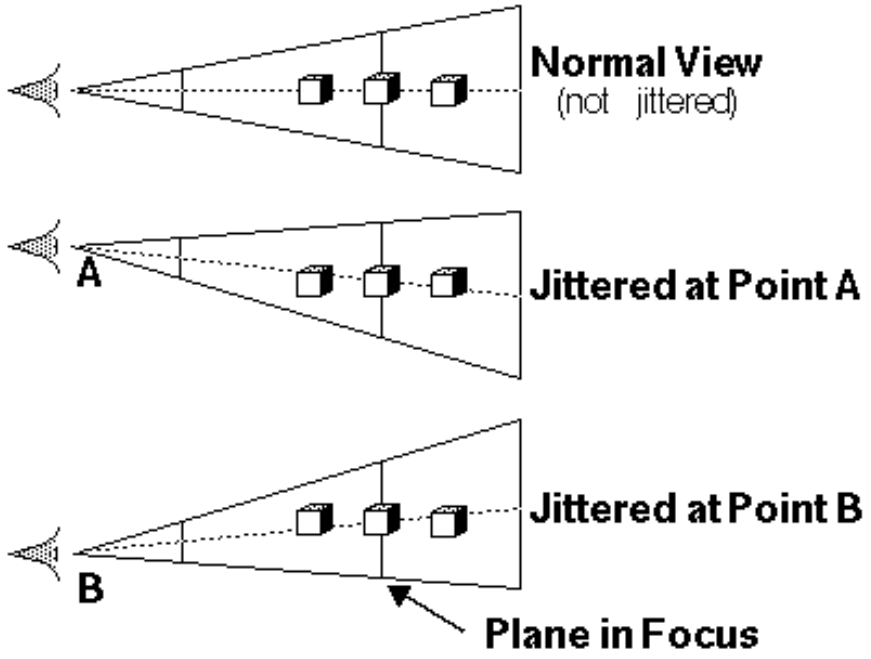
DoF – Ray tracing

- Cast rays across the lens
- Few samples → noise, artifacts
- Most accurate
- Slow



DoF – Accumulation Buffer

- Render scene multiple times from different locations
- Blend together using accumulation buffer
- Few samples
- Artifacts

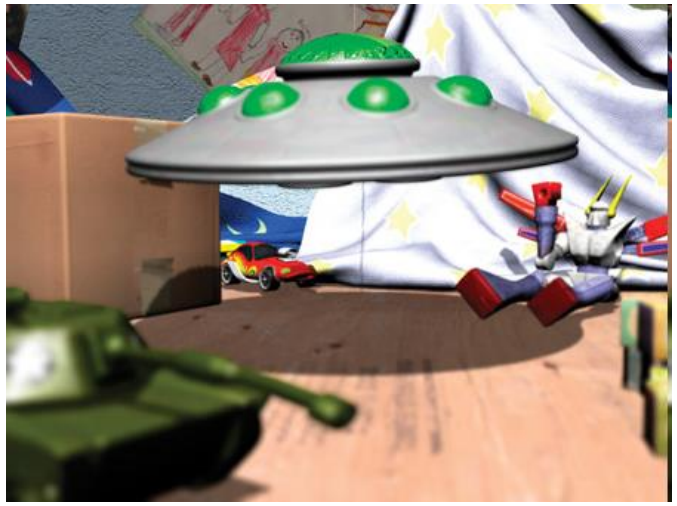


Layered DoF

- Objects sorted in layers
  - No overlapping in depth
  - Each layer blurred based on depth
  - Composition
- Problems
  - Large depth ranges
  - Overlapped objects
  - Blur factor

Forward-mapped Z-buffer

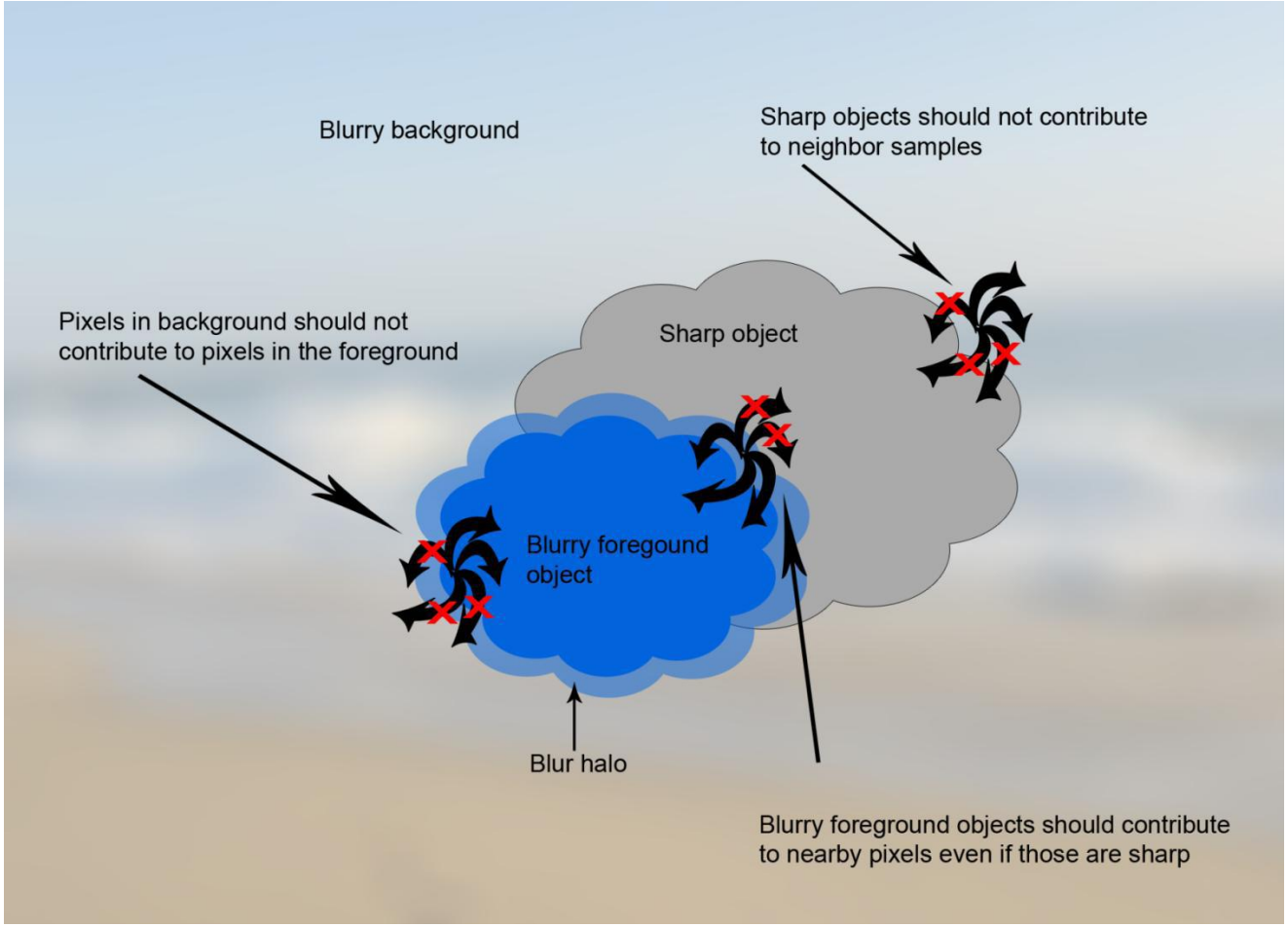
- Rendering sprites to approximate depth of field
  - Render scene to color and depth buffer
  - Use depth buffer to compute CoC for each pixel
  - Blend each pixel into framebuffer as circle with diameter equal to CoC and alpha inversely proportional to the circles' areas
  - Circles are blended only to pixels with higher depth
  - Renormalize pixels with alpha <> 1



Reverse-mapped Z-buffer

- Color buffer is blurred by varying amounts per pixel
  - Render scene to color and depth buffer
  - For each pixel, determine level of blurriness = difference between pixel depth and focus plane
  - Use several blurred and downsampled color buffers and combine them based on level of blurriness
  - Use blurred depth maps, CoC maps
  - Usage of mip-maps, render-to texture, Gaussian blur

DoF – SS approach

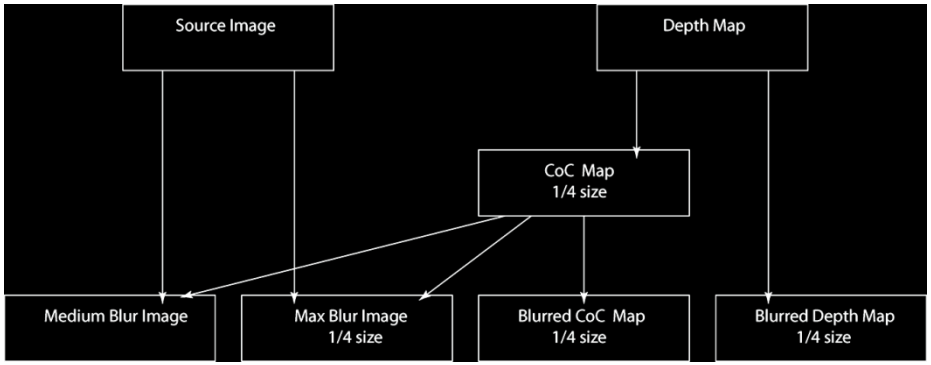


Blizzard

DoF – SS approach

- Generate three images for each level of blur
- Compute the CoC for each pixel, store it in CoC map
- Generated blurred CoC map and depth map
- Sample the source depth map and the blurred depth map and use depth ordering test to determine if the blurred or non-blurred CoC should be used
- Calculate contribution from each of the four blur sources based on the CoC factor and sum the contributions

CoC



$$saturate\left(\frac{DofAmount \times max(0, Depth - FocalDepth - NoBlurRange)}{MaxBlurRange - NoBlurRange}\right)$$

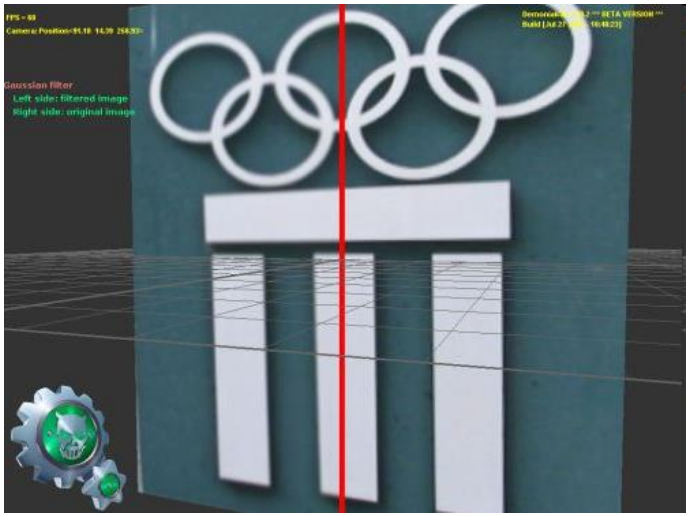
Blizzard

Gaussian blur - GLSL

- 2D Gaussian 3x3 filter
- Use separable approach ☺

```
varying vec2 vTexCoord;

// use shaders for rendering screen aligned quad
void main(void)
{
   gl_TexCoord[0] = gl_MultiTexCoord0;
   gl_Position = ftransform();
}
```




```
#define KERNEL_SIZE 9
// Gaussian kernel
// 1 2 1
// 2 4 2
// 1 2 1
const float kernel[KERNEL_SIZE] = { 1.0/16.0, 2.0/16.0, 1.0/16.0, 2.0/16.0, 4.0/16.0,
        2.0/16.0, 1.0/16.0, 2.0/16.0, 1.0/16.0 };

uniform sampler2D colorMap; // mapped color texture
uniform float width;  // width of mapped color texture
uniform float height; // height of mapped color texture

const float step_w = 1.0/width;
const float step_h = 1.0/height;
const vec2 offset[KERNEL_SIZE] = { vec2(-step_w, -step_h), vec2(0.0, -step_h),
        vec2(step_w, -step_h), vec2(-step_w, 0.0), vec2(0.0, 0.0), vec2(step_w, 0.0),
        vec2(-step_w, step_h), vec2(0.0, step_h), vec2(step_w, step_h) };

void main(void)
{
   int i = 0;
   vec4 sum = vec4(0.0);

   for( i=0; i<KERNEL_SIZE; i++ )  {
      vec4 tmp = texture2D(colorMap, gl_TexCoord[0].st + offset[i]);
      sum += tmp * kernel[i];
   }

   gl_FragColor = sum;
}
```

Gaussian blur - GLSL

- Separable approach – faster (9x9 filter)

```
float Weights[9] = {0.0162162162, 0.0540540541, 0.1216216216, 0.1945945946, 0.2270270270, 0.1945945946, 0.1216216216, 0.0540540541, 0.0162162162};
```

```
uniform float OffsetsH[9];
uniform float Weights[9];
uniform sampler2D Tex0;

void main (void)
{
   int i;
   vec4 color = vec4(0.0, 0.0, 0.0, 1.0);
   for(i=0; i<9; i++)
      color += (texture2D(Tex0, gl_TexCoord[0].st + vec2(OffsetsH[i], 0.0)) *Weights[i]);
   gl_FragColor = color;
}
```

Fragment shader for horizontal direction

```
for(int i=0; i<9; i++)
{
   OffsetsH[i] = (i - 4.0f) / float(Tex0Width);
   OffsetsV[i] = (i - 4.0f) / float(Tex0Height);
}
```

```
uniform float OffsetsV[9];
uniform float Weights[9];
uniform sampler2D Tex0;

void main (void)
{
   int i;
   vec4 color = vec4(0.0, 0.0, 0.0, 1.0);
   for(i=0; i<9; i++)
      color += (texture2D(Tex0, gl_TexCoord[0].st + vec2(0.0, OffsetsV[i])) *Weights[i]);
   gl_FragColor = color;
}
```

Fragment shader for vertical direction

Gaussian blur - GLSL

- Using bilinear filtering – doubling size of kernel – even faster (9x9 kernel)

http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/

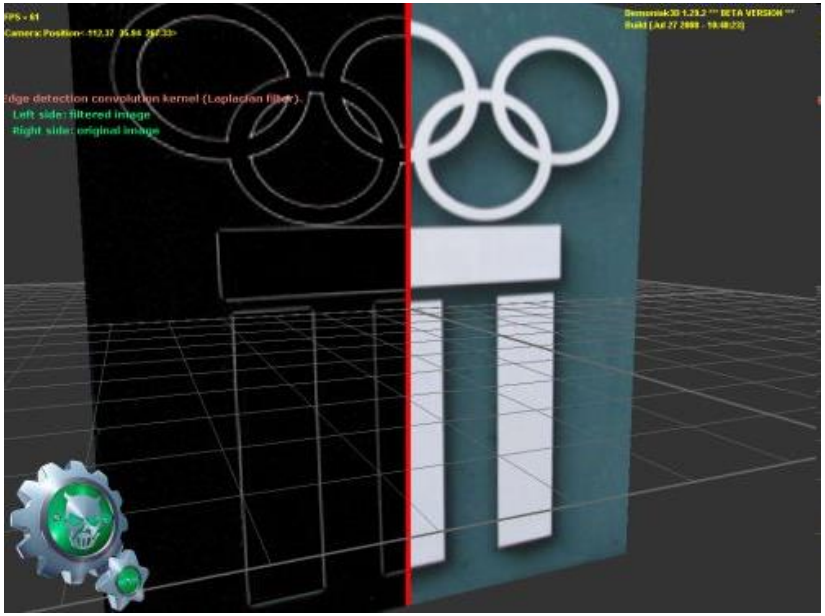
```
uniform sampler2D image;
uniform int image_height;
uniform float offset[3] = float[]( 0.0, 1.3846153846, 3.2307692308 );
uniform float weight[3] = float[]( 0.2270270270, 0.3162162162, 0.0702702703 );

void main(void)
{
    FragmentColor = texture2D( image, vec2(gl_FragCoord)/image_height ) * weight[0];
    for (int i=1; i<3; i++)
    {
        FragmentColor += texture2D( image, ( vec2(gl_FragCoord)+vec2(0.0, offset[i]) )/image_height ) * weight[i];
        FragmentColor += texture2D( image, ( vec2(gl_FragCoord)-vec2(0.0, offset[i]) )/image_height ) * weight[i];
    }
    gl_FragColor = FragmentColor;
}
```

Fragment shader for vertical direction

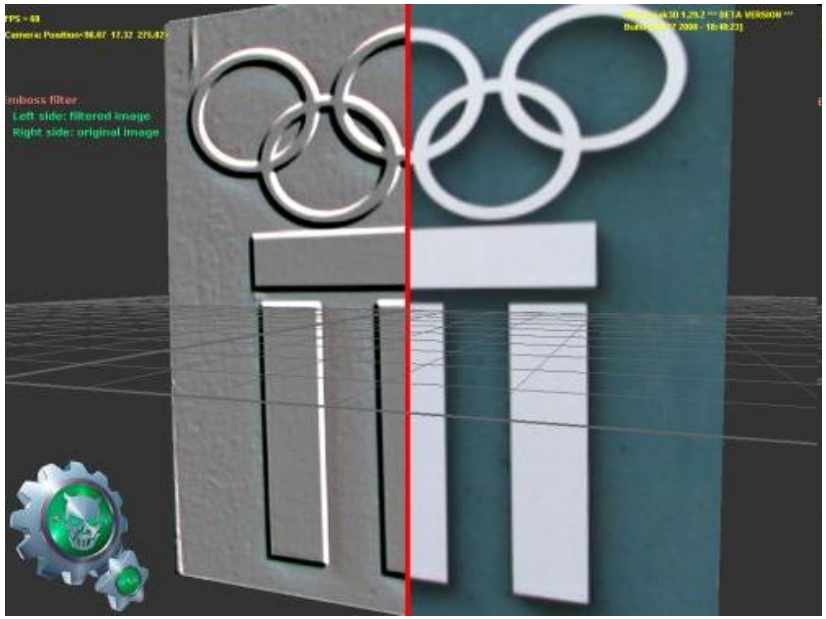
Other filters

http://www.ozone3d.net/tutorials/image_filtering.php



$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Laplacian Filter



$$\begin{matrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{matrix}$$

Emboss Filter

DoF – GLSL

Demo from http://encelo.netsons.org/programming/opengl

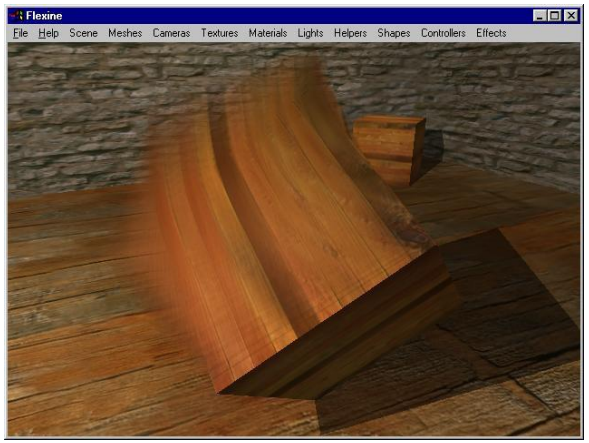
Motion blur

- Movement of an object - dynamics
- Blur - object / background
- Basic approach - precomputed
  - extended geometry (important parts)
  - various textures (wheels, road)

Accumulation buffer

- Average a series of images → accumulation buffer
- For each frame
  - Render frame
  - Store frame
  - Add frame to accumulation buffer
  - Subtract some older frame from accumulation buffer (from n back steps)
- Memory needed for previous frames
- Several previous frames can be generated in current frame – high frame rate necessary

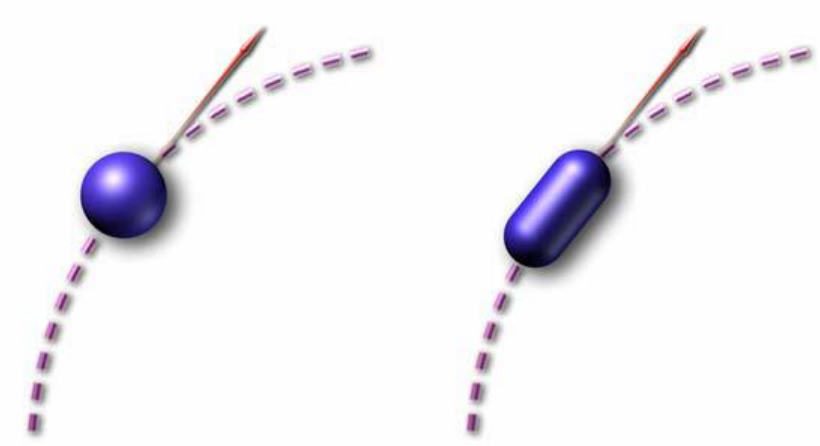
```
float q = .60;
glAccum(GL_MULT, q);
glAccum(GL_ACCUM, 1-q);
glAccum(GL_RETURN, 1.0);
```



Geometry distortion

- Drawing distorted object along path of movement, using different blending for various parts of distorted objects
- Vertex shader:
  - Distort each vertex in the velocity direction
  - Size of distortion is based on normal and velocity dot product
  - Add alpha based on size of distortion and velocity
  - Use velocity relative to camera
- Fragment shader
  - Use shading based on size of distortion

Geometry distortion



http://developer.amd.com/media/gpu_assets/ShaderX2_MotionBlurUsingGeometryAndShadingDistortion.pdf

Screen space

- Generate per-pixel scene velocity maps
- For movement of camera, can be extended to movement of objects
- Computation in fragment shader
  - Velocity (motion) vector is in screen space
  - We need previous position of fragment in screen space – it is calculated using previous frame matrix
  - Depth buffer value + current frame screen-space coordinates + inverse model-view-projection matrix = fragment object coordinates + prev. frame model-view-projection matrix = prev. frame screen-space coordinates

Screen space

- Post-processing effect
- Generating blur based on screen-space velocity
  - Sampling color buffer in velocity direction
  - Convolution
- No blur for some objects - masking
- Dynamic objects
  - Masking parts of scene where is dynamic object
  - Storing screen-space velocities of object

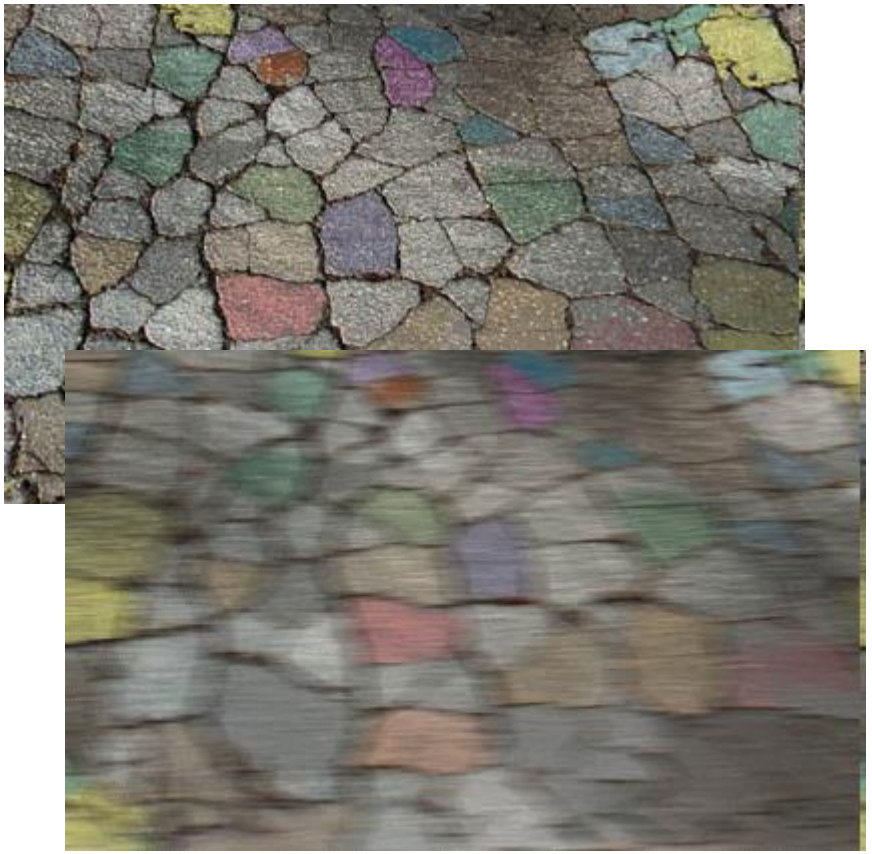
Motion blur – GLSL

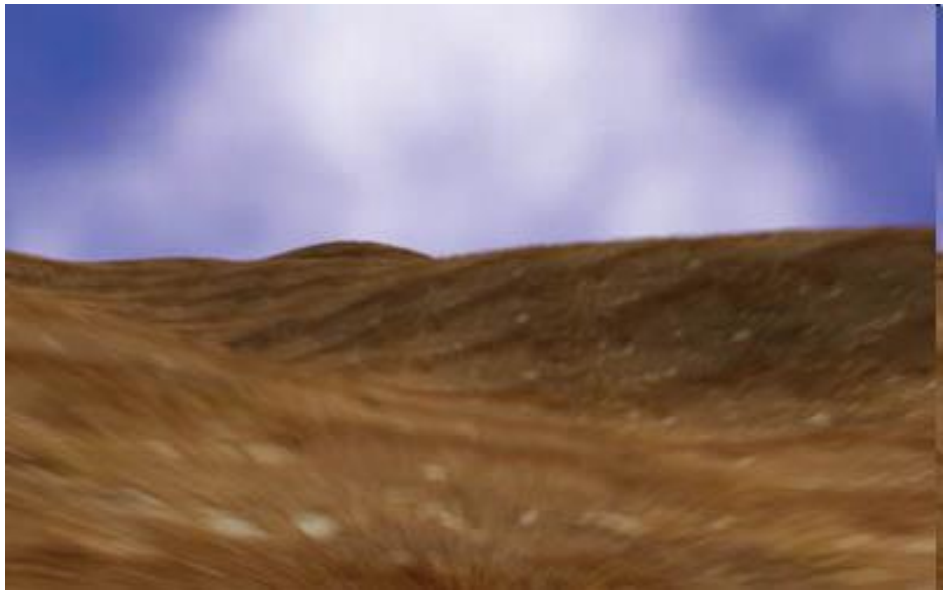
Rendering full screen quad

```
uniform sampler2D colorMap; // mapped color texture
uniform sampler2D depthMap; // mapped color texture
uniform mat4 previousModelViewProjectionMatrix;
uniform int numSamples;

void main(void)
{
  // Get the depth buffer value at this pixel.
  float zOverW = shadow2D(depthTexture, vec3(gl_TexCoord[0])).r;
  // current screen-space position at this pixel in the range -1 to 1.
  vec4 currentPos = vec4(gl_TexCoord[0].s * 2 - 1, (1 - gl_TexCoord[0].t) * 2 - 1,  zOverW, 1);
  // Transform by the model-view-projection inverse.
  vec4 D = gl_ModelViewProjectionMatrixInverse * currentPos;
  // Divide by w to get the object space position.
  vec4 worldPos = D / D.w;
  // Use the world position, and transform by the previous model-view-projection matrix.
  vec4 previousPos = previousModelViewProjectionMatrix * worldPos;
  // Convert to nonhomogeneous points [-1,1] by dividing by w.
  previousPos /= previousPos.w;
  // Use this frame's position and last frame's to compute the pixel velocity.
  vec2 velocity = (currentPos - previousPos)/2.f;
  // Get the initial color at this pixel.
  vec4 color = texture2D(colorMap, gl_TexCoord[0].st);
  vec2 texCoord = gl_TexCoord[0].st + velocity;
  for (int i = 1; i < numSamples; ++i, texCoord += velocity)
  {
    // Sample the color buffer along the velocity vector and it to color sum
    color += texture2D(colorMap, texCoord);
  }
  // Average all of the samples to get the final blur color.
  gl_FragColor = color / numSamples;
}
```

Motion blur

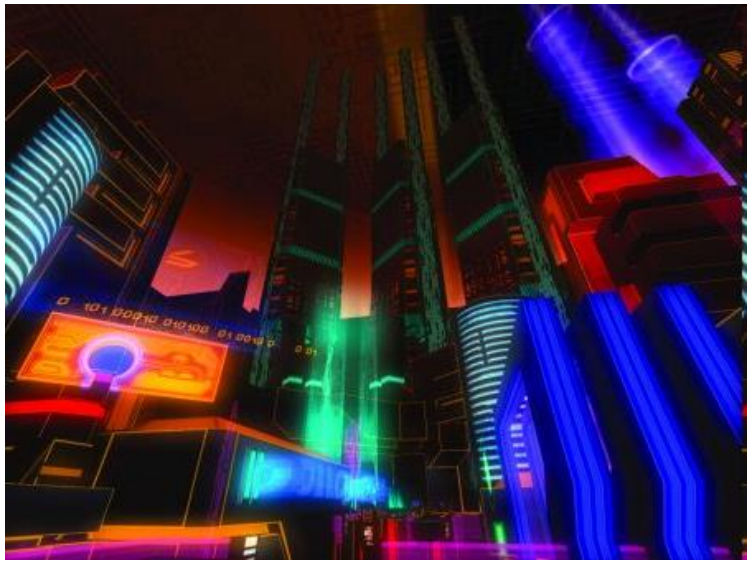




http://http.developer.nvidia.com/GPUGems3/gpugems3_ch27.html

Glow

- Glow / Bloom / Glare add visual cues about brightness and atmosphere in scene
- Reproducing the visual effects of intense light, very bright sources

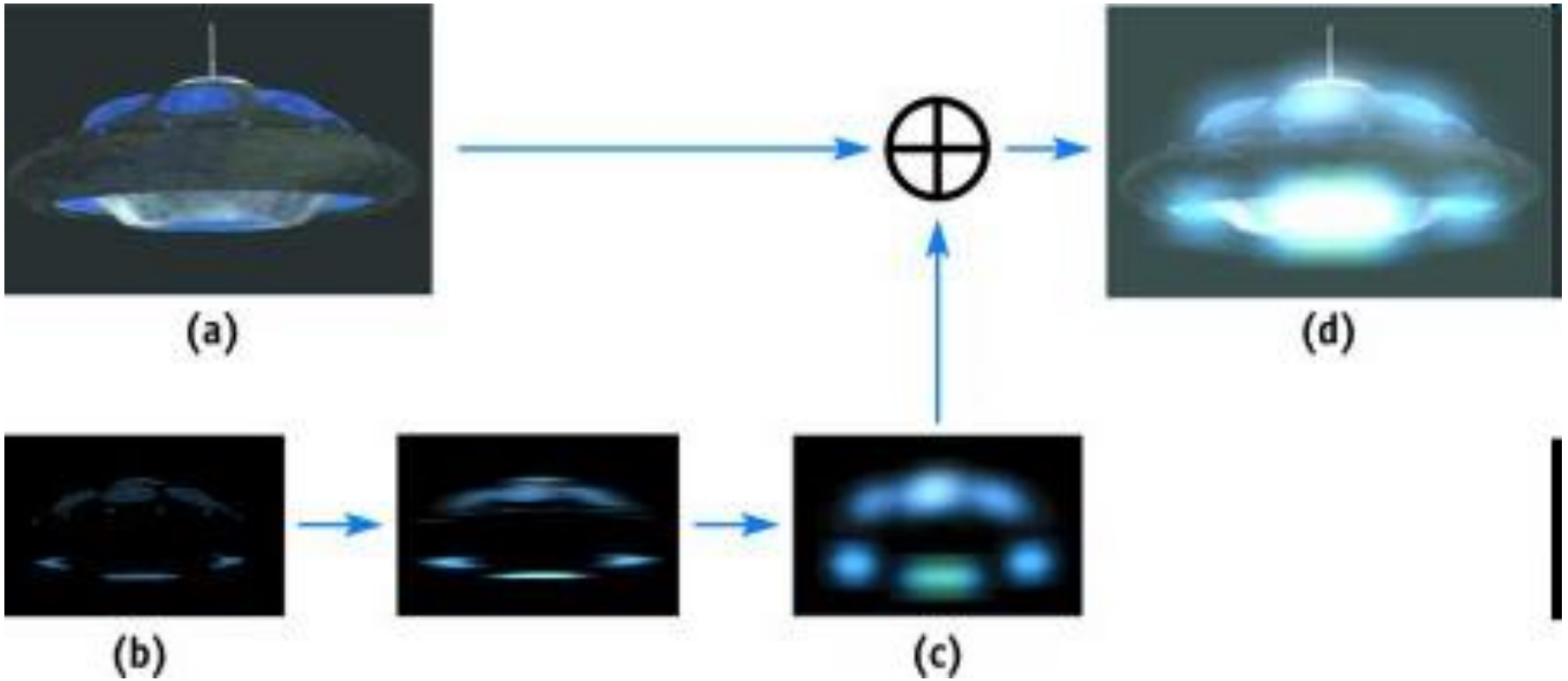


Tron 2.0

Glow

- Simple – use billboards and blending to put bright mask over bright sources
- Use two pass rendering to combine normal scene with glowing parts of screen
  - Render scene to texture, save glow parameter in alpha or downsampled texture
  - Threshold small values of glow
  - Use filter to blur glow values
  - Use additive blending to combine glow values with scene texture pixels

Glow



http://www.gamasutra.com/view/feature/2107/realtime_glow.php

High Dynamic Range

- RGB images – small range of tonal values, can't reproduce high luminance range between lightest and darkest areas, 8 bits per channel – 256:1
- Real-world luminance dynamic range - 100000:1
- HDR rendering – luminance and radiance per pixel exceeds range [0.0, 1.0] - Computation in floating point format

Lum = 0.3*R + 0.59*G + 0.11*B

Far Cry

HDR sources

- CG rendering
- Photography – different exposure times
  - Different details visible at low and high exposure

Dean S. Pemberton

HDR sources

Dean S. Pemberton

HDR requirements

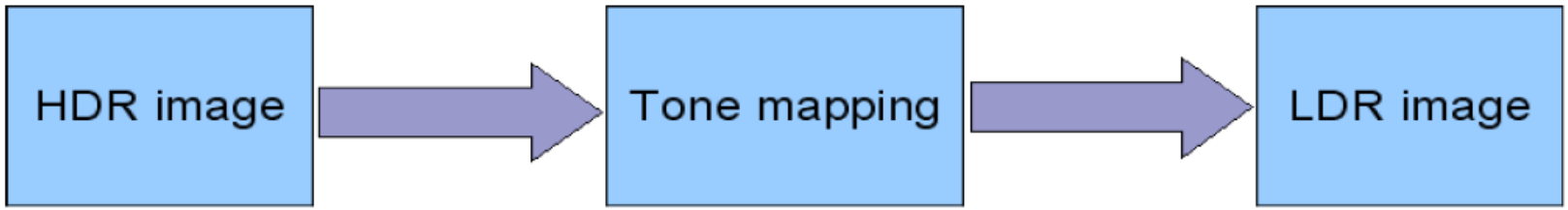
- All lightning, post-processing, texturing, … calculations must be in floating points
- Floating-point arithmetic in shaders
- Channels of used textures must exceed [0,1] – floating point textures
  - Texture internal format GL_RGBA16F, GL_RGBA32F
- Rendering to texture = floating point render targets with blending
  - Texture color attachments to FBO with internal format GL_RGBA16F, GL_RGBA32F

HDR Maps

- Radiance
  - Greg Ward, 1985
  - RGBE file format, 32bpp, shared exponent
  - http://www.graphics.cornell.edu/~bjw/rgbe.html
- OpenEXR
  - Industrial Light & Magic, 2003
  - Open standard, robust
  - http://www.openexr.com/

Tone mapping

- Mapping HDR source to LDR destination (image, display, …), mapping [0,inf) to [0,1)
- Global
  - For every mapped pixel takes into account the intensity of whole image
- Local
  - For every mapped pixel takes into account surrounding pixel intensities



Tone mapping

- Simple $$Lum_{mapped}(F) = \frac{Lum(F)}{Lum(F)+1}$$
- Using average luminance of whole image
  - Given as parameter
  - Convert RGB frame to luminance and downscale texture using bilinear filtering to 1x1 – average luminance

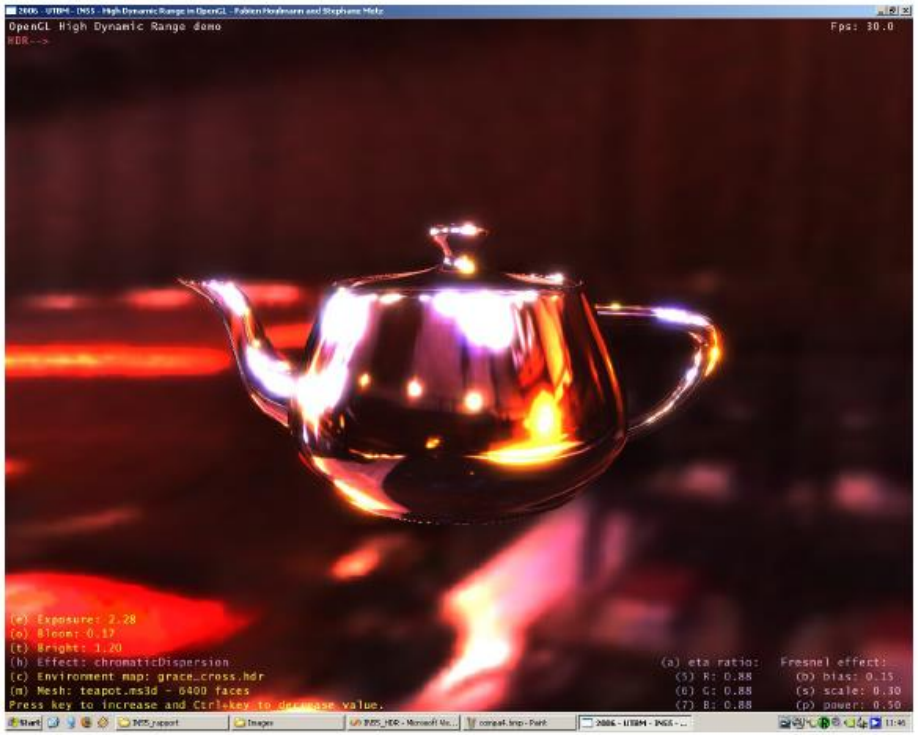
$$Lum_{mapped}(F) = \alpha * \frac{Lum(F)}{Lum_{average}}$$

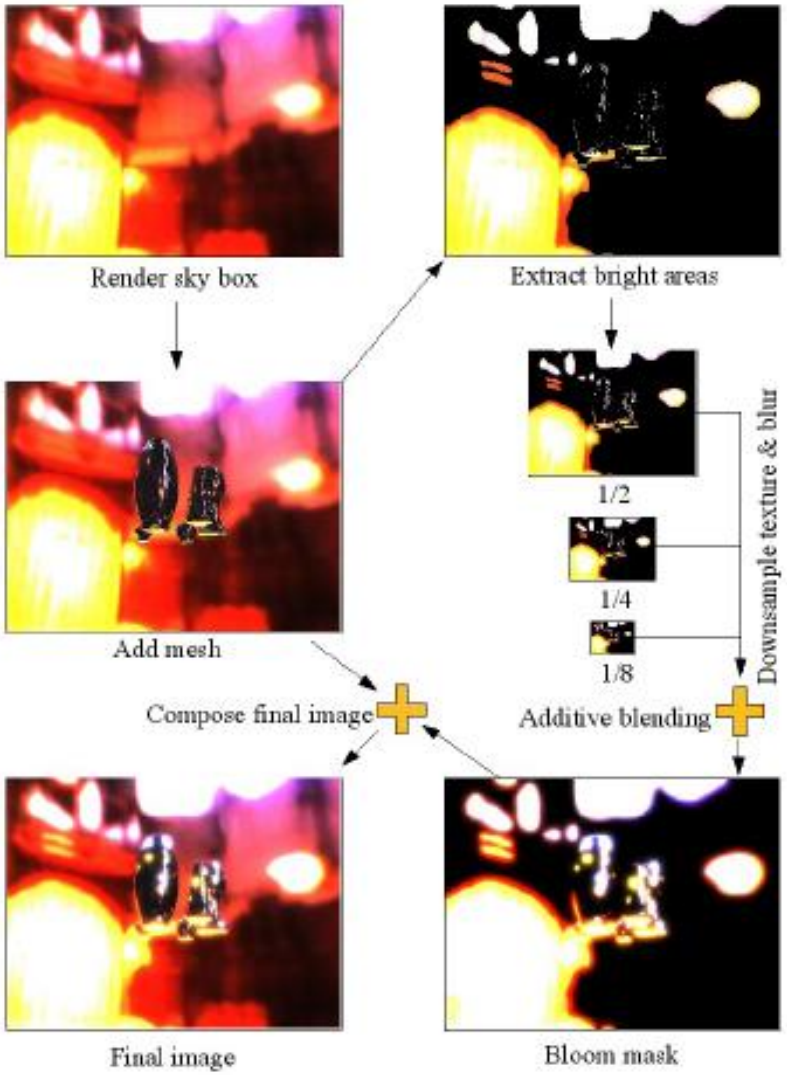
$$Lum_{mapped}(F) - final\,(LDR)\,luminance\,of\ fragment\,F$$
$$Lum_{average} - average\,luminance\,of\,the\,whole\,frame$$
$$Lum(F) - Source\,(HDR)\,luminance\,of\ fragment\,F$$

- Using maximal luminance and exposure

$$Lum_{mapped}(F) = Lum(F) * exposure * \frac{1+\frac{exposure}{Lum_{max}}}{1+exposure}$$

HDR + bloom





Real-time Graphics
Martin Samuelčík

HDR – OpenGL

Configuring FBO for rendering to floating point texture

```
GLint frameBuffer, depthBuffer, frame;
glGenFramebuffersEXT(1, &frameBuffer);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, frameBuffer);
glGenRenderbuffersEXT(1, &depthBuffer);
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depthBuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT, windowWidth, windowHeight);
glGenTextures(1, &frame);
glBindTexture(GL_TEXTURE_2D, frame);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB16F_ARB,  windowWidth, windowHeight, 0, GL_RGB, GL_FLOAT, NULL);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, frame, 0);
```

Creating floating point texture

```
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
float* data = LoadData("texture.hdr");
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB16F_ARB,  textureWidth, textureHeight, 0, GL_RGB, GL_FLOAT, data);
GLint format;
glGetTexLevelParameteriv(GL_TEXTURE_2D, 0, GL_TEXTURE_INTERNAL_FORMAT, &format);
```

HDR – GLSL

```
// TONE MAPPING FRAGMENT SHADER
// render texture and bloom map
uniform sampler2D tex, bloom;
// Control exposure with this value
uniform float exposure;
// How much bloom to add
uniform float bloomFactor;
// Max luminance
uniform float brightMax;

void main()
{
   vec2 st = gl_TexCoord[0].st;
   vec4 color = texture2D(tex, st);
   vec4 colorBloom = texture2D(bloom, st);
   // Add bloom to the image
   color += colorBloom * bloomFactor;
   // Perform tone-mapping
   float YD = exposure * (exposure/brightMax + 1.0);
   YD = YD / (exposure + 1.0);
   color *= YD;
   gl_FragColor = color;
}
```

```
// ANOTHER TONE MAPPING FRAGMENT SHADER
varying vec3 texCoordinate;
uniform sampler2D frame;
uniform float exposure;

void main()
{
   vec4 colorTemp = texture2D(frame, texCoordinate.xy);
   vec4 color = exposure*colorTemp;
   float temp  = (pow(exposure, 0.7))/(pow(exposure, 0.7)+pow(20.0, 0.7));
   gl_FragColor = color * temp;
}
```

```
// ILLUMINATION FRAGMENT SHADER USING HDR CUBE TEXTURES
uniform samplerCube hdrTexDiffuseSampler;
uniform samplerCube hdrTexReflectionSampler;
// interpolated normal vector from vertex shader
varying vec3 N;
// interpolated reflect vector from vertex shader
varying vec3 R;

void main (void)
{
   // mix material color, diffuse cube map and reflection cube map
   vec4 matColor = vec4(0.7, 0.7, 0.7, 1.0);
   vec4 diffuse = mix(matColor, textureCube(hdrTexDiffuseSampler, N), 0.9);
   vec4 reflected = textureCube(hdrTexReflectionSampler, R);
   vec4 color = mix(diffuse, reflected, 0.4);
   color.a = 1.0;
   gl_FragColor = color;
}
```

Sources

- http://developer.amd.com/media/gpu_assets/ShaderX2_Real-TimeDepthOfFieldSimulation.pdf
- http://http.developer.nvidia.com/GPUGems3/gpugems3_ch28.html
- http://http.developer.nvidia.com/GPUGems/gpugems_ch23.html
- http://encelo.netsons.org/programming/opengl
- http://http.developer.nvidia.com/GPUGems3/gpugems3_ch27.html
- http://www.gamasutra.com/view/feature/2107/realtime_glow.php
- http://www.gamerendering.com/2008/10/11/gaussian-blur-filter-shader/
- http://code.google.com/p/transporter-game/downloads/detail?name=HDRRenderingInOpenGL.pdf
- http://http.download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_HDR.pdf
- http://www.spieleprogrammierung.net/

Questions?