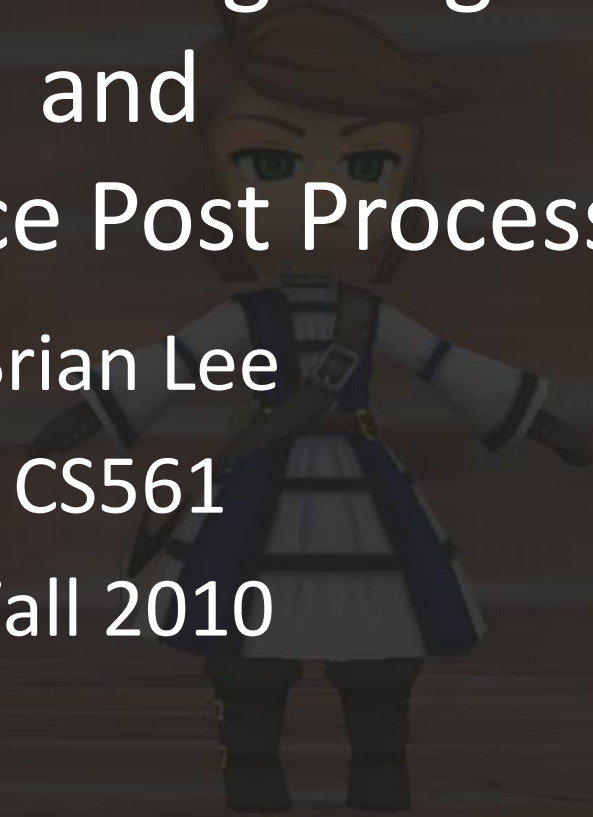


Deferred Lighting and Screen-Space Post Processing

Brian Lee

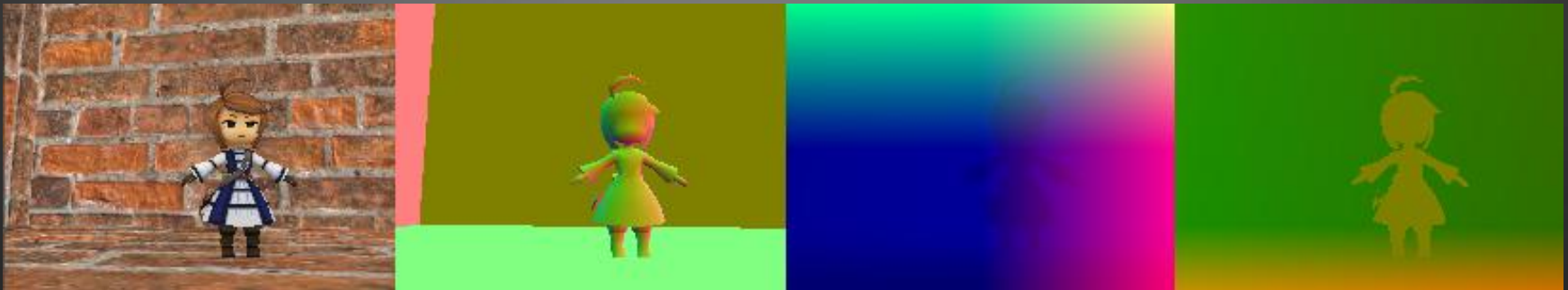
CS561

Fall 2010



Geometry Buffer (G-Buffer)

- Using Multiple Render Targets (MRT), newer graphics cards can output geometry and other information to several textures with one draw call
- The G-Buffer can later be sampled while drawing a full-screen quad, with lighting and other calculations being performed in screen space.



Diffuse Color

Normals (World)

Position (Screen)

Velocity (Screen)

Complexity

- One of the major benefits of deferred lighting is that lighting calculations are done only on visible lit pixels after all geometry transforms are done.
 - This reduces the complexity from $O(\#lights * \#objects)$ to $O(\#lights * screen\ area[constant])$
- This is beneficial for scenes with many small point/spot lights and few large/directional lights.

Drawbacks

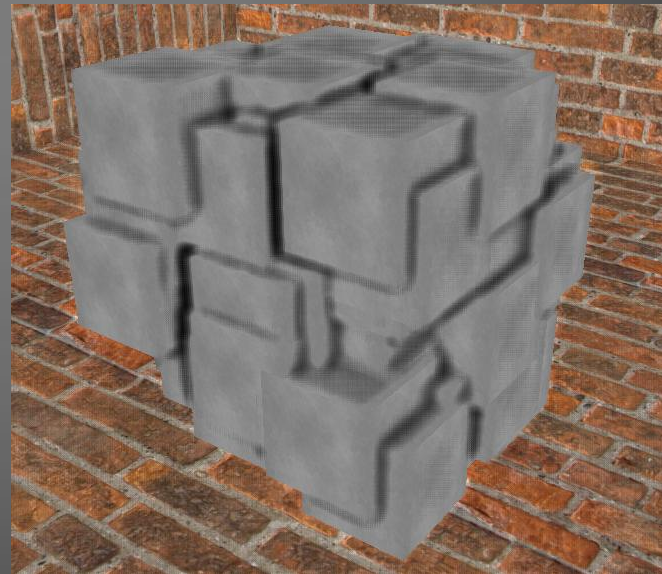
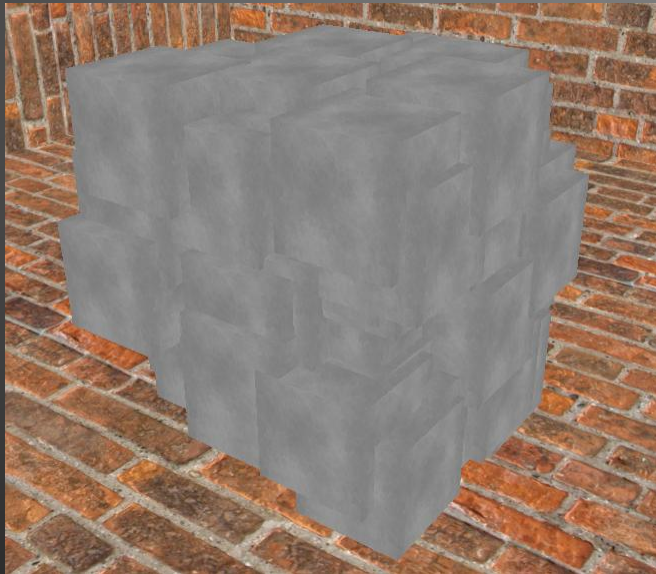
- Limited render targets for packing data
- Normalizing large ranges to 8 or 16 bit floats reduces precision.
- Transparency is difficult to implement and still very limited. Drawing transparent objects in separate passes is more feasible.

Other Techniques...

- Although screen space lighting calculation is useful, having a deferred pipeline with a G-Buffer is an important first step for a number of post-processing techniques, such as...

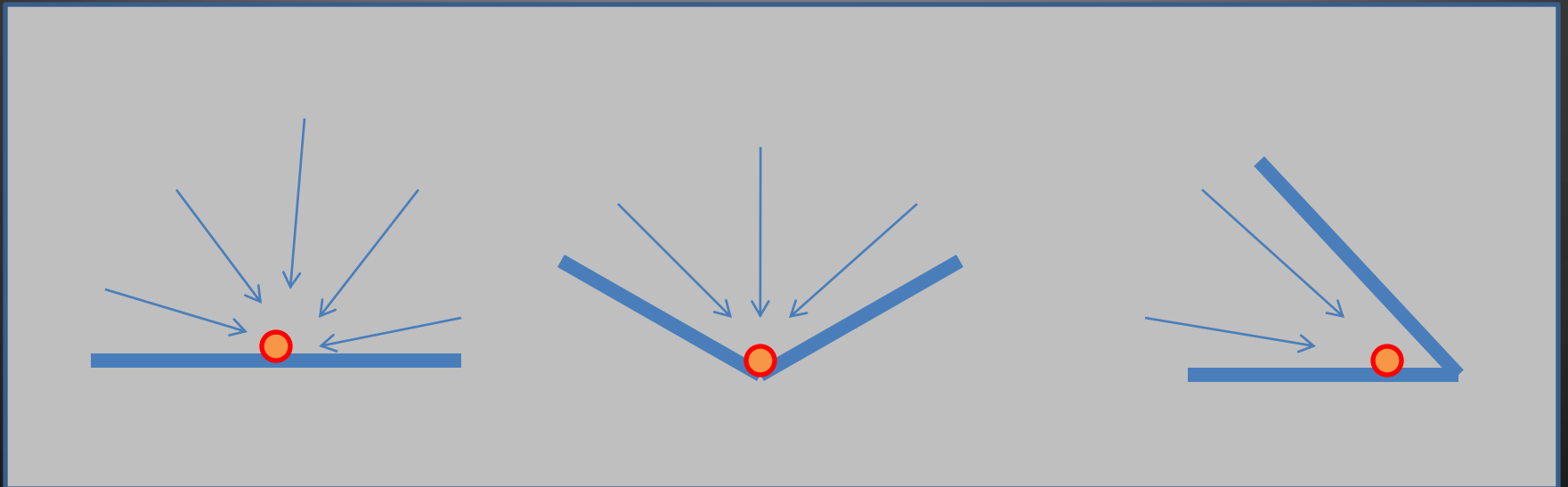
Screen Space Ambient Occlusion (SSAO)

- SSAO uses position and normal data from the G-Buffer to calculate ambient occlusion in screen space, enhancing contours and crevices in the scene.



Ambient Occlusion

- Ambient light is uniform, simulating light reflected from all directions
- Ambient occlusion is when points on a surface have their ambient light partially occluded by the surrounding geometry, as seen below.
- Traditionally, Ambient Occlusion is computed by casting rays from each point and intersecting with the surrounding geometry in world space, but this is not practical for real time rendering.

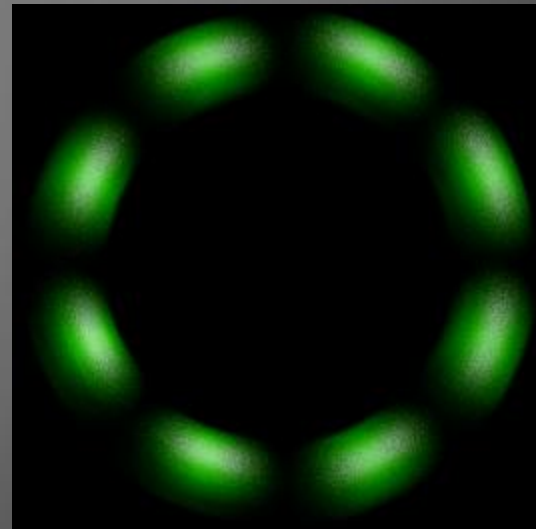
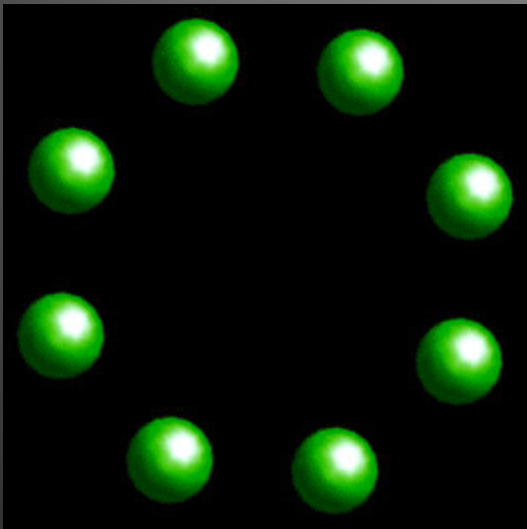


In Screen Space...

- Get pixel position/normal from g-buffers
- Cast rays randomly (using a noise texture) in direction of normal from pixel
- Test samples against depth buffer, if they fall behind it, the pixel is more occluded
- Filter the resulting SSAO buffer (falloff, blur)

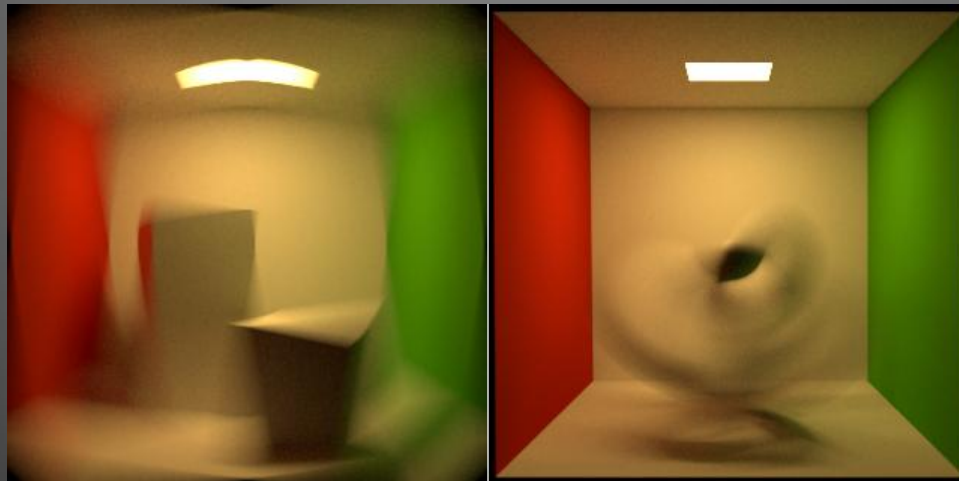
Motion Blur

- The Human visual system perceives a moving blurred object as a fast-moving sharp object.
- In computer graphics, large movements between frames can be jarring
- Motion blur minimizes this effect and can make even low frame rate animation look smooth.



Motion Blur Methods

- Multisampling
 - Render objects multiple times per frame and average
 - Quality depends on number of samples, very slow



Multisampled ray traced motion blur. High quality, extremely slow.

Motion Blur Methods

- GPU Gems 3 – “Motion Blur as a Post-Processing Effect”
 - Extract each pixel’s world space position from depth buffer
 - Use previous frame’s View/Projection Matrix to calculate pixel velocity
 - Average pixel with others along velocity vector
 - Only requires depth buffer access
 - Only works with camera movement, not dynamic objects



Motion Blur Methods

- Same method, enhanced with G-Buffer:
 - For each object, use previous frame's WVP matrix to store per-pixel screen space velocity vectors in a texture.
 - Average colors along velocity vectors in the same manner as before
 - Works with dynamic objects



Screen Space Velocity Buffer with model rotating clockwise
Normalized [0,1]

Motion Blur Methods

- Enhanced GPU Gems Method Cons
 - Weird artifacts with object blurring, focal objects must be masked out



Stationary camera with rotating model



Camera rotating relative to model (focal object)

My Motion Blur Method

Camera rotating relative to model (focal object)

GPU Gems Method



My Method



(No object masking required!)

My Motion Blur Method

Stationary camera with rotating model

GPU Gems Method



My Method



(No object masking required!)

My Motion Blur Method

- Similar to a depth of field effect, except with a directional blur based on velocity, rather than Gaussian blur based on depth.
- Each pixel is averaged with neighboring pixels within a certain radius, but only if it lies on (or very near) the neighboring pixel's velocity vector.

My Motion Blur Method

```
for each screen space pixel [center pixel]
{
    sample center pixel depth and color from g-buffer;
    set successes = 0
    //sample in a radius around center pixel
    for(y = -radius; y < radius; ++y)
    {
        for(x = -radius; x < radius; ++x)
        {
            sample depth of test pixel from g-buffer
            //check if test pixel is deeper than center pixel
            //if so, don't add it to center pixel!
            if(sample depth - center depth < EPSILON)
            {
                sample velocity of test pixel from g-buffer
                //get distance from center pixel to test pixel's screen space velocity vector
                difference = [test pixel].xy - [center pixel].xy
                dist = abs( dot( normalize( difference ) , normalize ( [center pixel].xy ) ) )
                //if center pixel is on or very close to test pixel's velocity line...
                if(dist > (1-EPSILON))
                {
                    //check if center pixel is within the velocity vector's magnitude
                    // (in either direction)
                    if(length(difference) >= length(velocity))
                    {
                        //if so, add test pixel's color to center pixel
                        [center pixel].rgb += [test pixel].rgb;
                        ++successes
                    }
                }
            }
        }
    }
}
//average center pixel color by number of successes
[center pixel].rgb /= successes
}
```

Pros/Cons

- Pros:
 - Depth check eliminates deeper objects bleeding into shallower objects.
 - No need for object masking for static or dynamic objects.
- Cons:
 - Computationally more expensive:
NxN square filter vs. N linear filter
 - Blur size limited by filter radius/step size

Enhancements?

- DX10+, Shader 4+
- Vary filter radius/step size based on depth
- Vary filter parameters based on camera velocity

Demo time!

References:

Deferred-

Valient, Michal. 2007. "Deferred Rendering in Killzone 2"

http://www.guerrilla-games.com/publications/dr_kz2_rsx_dev07.pdf

SSAO-

Shanmugam, Perumaal. Arikan, Okan. "Hardware Accelerated Ambient Occlusion Techniques on GPUs"

<https://faculty.digipen.edu/~gherron/references/References/GlobalLighting/ao.pdf>

Sainz, Miguel. 2008. "Real-Time Depth Buffer Based Ambient Occlusion"

http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_Ambient_Occlusion.pdf

Motion Blur-

Rosado, Gilberto. GPU Gems 3 "Motion Blur as a Post-Processing Effect" -

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

Gilham, David. 2006. "Real-Time Depth-of-Field Implemented with a Post-Processing Only Technique." In *Shader X5*, edited by Wolfgang Engel, pp. 163–175. Charles River Media.

Knight model courtesy of Max Hancock - <http://relaxingdog.com/>