

CSE 167:
Introduction to Computer Graphics
Lecture #17: Deferred Rendering

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2019

Announcements

- ▶ TA evaluations
- ▶ CAPE evaluation

Lecture Overview

- ▶ Deferred Rendering
 - ▶ Deferred Shading
 - ▶ Bloom and Glow
 - ▶ Screen Space Ambient Occlusion

Deferred Rendering

- ▶ Opposite to Forward Rendering, which is the way we have rendered with OpenGL so far
- ▶ Deferred rendering describes post-processing algorithms
 - ▶ Requires two-pass rendering
 - ▶ First pass:
 - ▶ Scene is rendered as usual by projecting 3D primitives to 2D screen space.
 - ▶ Additionally, an off-screen buffer (G-buffer) is populated with additional information about the geometry elements at every pixel
 - Examples: normals, diffuse shading color, position, texture coordinates
 - ▶ Second pass:
 - ▶ An algorithm, typically implemented as a shader, processes the G-buffer to generate the final image in the back buffer

Deferred Shading

- ▶ Postpones shading calculations for a fragment until its visibility is completely determined
 - ▶ Only visible fragments are shaded
- ▶ Algorithm:
 - ▶ Fill a set of buffers with common data, such as diffuse texture, normals, material properties
 - ▶ Render lights with limited extent and use data from the buffers for the lighting computation
- ▶ Advantages:
 - ▶ Decouples lighting from geometry rendering
 - ▶ Several lights can be applied with a single draw call. E.g., >1000 lights can be rendered at 60 fps
- ▶ Disadvantages:
 - ▶ More expensive (memory, bandwidth, shader instructions)
- ▶ Tutorial:
 - ▶ <http://gamedevs.org/uploads/deferred-shading-tutorial.pdf>



*Particle system with glowing particles.
Source: Humus 3D*

Deferred Lighting

▶ Video:

▶ <https://www.youtube.com/watch?v=zOVsxldANcg>



Lecture Overview

- ▶ Deferred Rendering Techniques
 - ▶ Deferred Shading
 - ▶ **Bloom and Glow**
 - ▶ Screen Space Ambient Occlusion

Bloom Effect

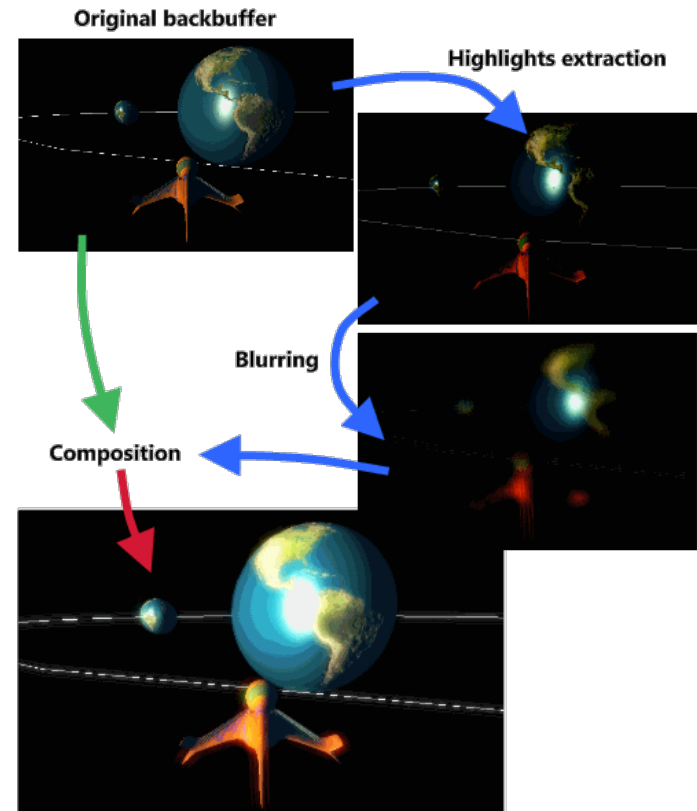


Left: no bloom, right: bloom. Source: <http://jmonkeyengine.org>

- ▶ Computer displays have limited dynamic range
- ▶ Bloom gives a scene a look of bright lighting and overexposure
- ▶ Provides visual cues about brightness and atmosphere
 - ▶ Caused by light scattering in atmosphere, or within our eyes

Bloom Shader

- ▶ **Step 1:** Extract all highlights of the rendered scene, superimpose them and make them more intense
 - ▶ Operates on G-buffer
 - ▶ Often done with G-buffer smaller (lower resolution) than frame buffer
 - ▶ Highlights found by thresholding luminance
- ▶ **Step 2:** Blur off-screen buffer, e.g., using Gaussian blur
- ▶ **Step 3:** Composite off-screen buffer with back buffer



*Bloom shader render steps.
Source: <http://www.klopfenstein.net>*

Glow vs. Bloom

- ▶ Bloom filter looks for highlights automatically, based on a threshold value
- ▶ If you want to have more control over what glows and does not glow, a glow filter is needed
- ▶ Glow filter adds an additional step to Bloom filter: instead of thresholding, only the glowing objects are rendered
- ▶ Render passes:
 - ▶ Render entire scene back buffer
 - ▶ Render only glowing objects to a smaller off-screen glow buffer
 - ▶ Apply a bloom pixel shader to glow buffer
 - ▶ Compose back buffer and glow buffer together

Video: Glowing Lava

- ▶ <https://www.youtube.com/watch?v=hmsMk-skqul>



References

- ▶ **Bloom Tutorial**

- ▶ <http://prideout.net/archive/bloom/>

- ▶ **GPU Gems Chapter on Glow**

- ▶ http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch21.html

- ▶ **GLSL Shader for Gaussian Blur**

- ▶ http://www.ozone3d.net/tutorials/image_filtering_p2.php

Lecture Overview

- ▶ Deferred Rendering Techniques
 - ▶ Deferred Shading
 - ▶ Glow
 - ▶ Screen Space Ambient Occlusion

Screen Space Ambient Occlusion (SSAO)

- ▶ “Screen Space” → deferred rendering approach
- ▶ Approximates ambient occlusion in real time
- ▶ Developed by Vladimir Kajalin (Crytek)
- ▶ First use in PC game Crysis (2007)



SSAO component

Ambient Occlusion

- ▶ Crude approximation of global illumination
- ▶ Often referred to as "sky light"
- ▶ Global method (not local like Phong shading)
 - ▶ Illumination at each point is a function of other geometry in the scene
- ▶ Appearance is similar to what objects appear as on an overcast day
 - ▶ Assumption: concave objects are hit by less light than convex ones

Basic SSAO Algorithm

- ▶ **First pass:**
 - ▶ Render scene normally and write z values to G-buffer's alpha channel
- ▶ **Second pass:**
 - ▶ Pixel shader samples depth values around the processed fragment and computes amount of occlusion, stores result in red channel
 - ▶ Occlusion depends on depth difference between sampled fragment and currently processed fragment



Ambient occlusion values in red color channel
Source: www.gamereading.com

SSAO With Normals

- ▶ **First pass:**
 - ▶ Render scene normally and copy z values to G-buffer's alpha channel and scene normals to RGB channels
- ▶ **Second pass:**
 - ▶ Use normals and z-values to compute occlusion between current pixel and several samples around that pixel



No SSAO



With SSAO

SSAO Discussion

▶ Advantages:

- ▶ Deferred rendering algorithm: independent of scene complexity
- ▶ No pre-processing, no memory allocation in RAM
- ▶ Works with dynamic scenes
- ▶ Works in the same way for every pixel
- ▶ No CPU usage: executed completely on GPU

▶ Disadvantages:

- ▶ Local and view-dependent (dependent on adjacent texel depths)
- ▶ Hard to correctly smooth/blur out noise without interfering with depth discontinuities, such as object edges, which should not be smoothed out

SSAO References

- ▶ **Nvidia's documentation**

- ▶ <http://developer.download.nvidia.com/SDK/10.5/direct3d/Source/ScreenSpaceAO/doc/ScreenSpaceAO.pdf>

Other Deferred Rendering Effects

- ▶ Demo: ReShade
 - ▶ <https://reshade.me>
 - ▶ Needs compatible app to run with

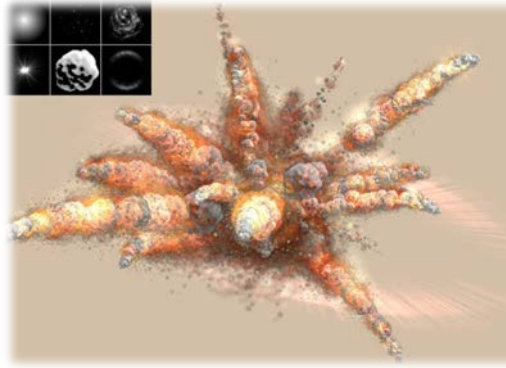
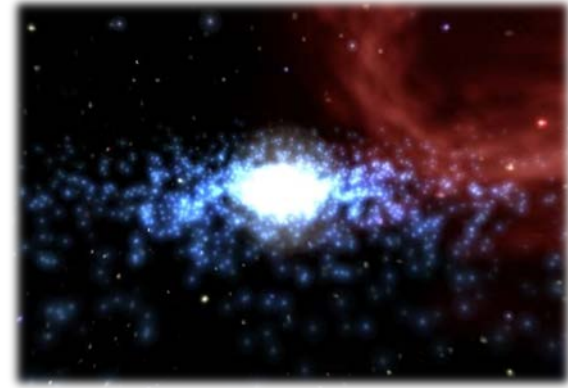
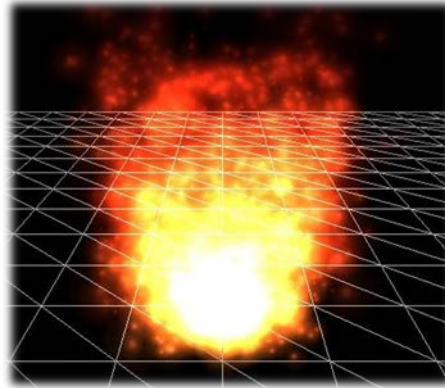
Lecture Overview

- ▶ Particle Systems
- ▶ Collision Detection
- ▶ Bump Mapping

Particle Systems

Particle Systems

- ▶ Used for:
 - ▶ Fire/sparks
 - ▶ Rain/snow
 - ▶ Water spray
 - ▶ Explosions
 - ▶ Galaxies

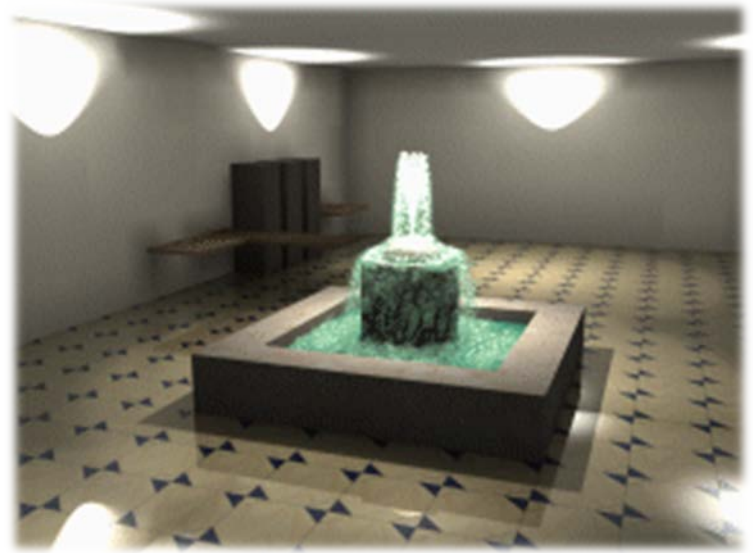


Internal Representation

- ▶ Particle system is collection of a number of individual elements (particles)
 - ▶ Controls a set of particles which act autonomously but share some common attributes
- ▶ Particle Emitter: Source of all new particles
 - ▶ 3D point
 - ▶ Polygon mesh: particles' initial velocity vector is normal to surface
- ▶ Particle attributes:
 - ▶ position (3D)
 - ▶ velocity (vector: speed and direction)
 - ▶ color + opacity
 - ▶ lifetime
 - ▶ size
 - ▶ shape
 - ▶ weight

Dynamic Updates

- ▶ Particles change position and/or attributes with time
- ▶ Initial particle attributes often created with random numbers
- ▶ Frame update:
 - ▶ Parameters: simulation of particles, can include collisions with geometry
 - ▶ Forces (gravity, wind, etc) accelerate a particle
 - ▶ Acceleration changes velocity
 - ▶ Velocity changes position
 - ▶ Rendering:
 - ▶ GL_POINTS
 - ▶ GL_POINT_SPRITE
 - ▶ Point shader



Source: <http://www.particlesystems.org/>

Point Rendering – Vertex Shader

```
uniform mat4 u_MVPMatrix;
uniform vec3 u_cameraPos;

// Constants (tweakable):
const float minPointSize = 0.1;
const float maxPointSize = 0.7;
const float maxDistance = 100.0;

void main()
{
    // Calculate point scale based on distance from the viewer
    // to compensate for the fact that gl_PointSize is the point
    // size in rasterized points / pixels.
    float cameraDist = distance(a_position_size.xyz, u_cameraPos);
    float pointScale = 1.0 - (cameraDist / maxDistance);
    pointScale = max(pointScale, minPointSize);
    pointScale = min(pointScale, maxPointSize);

    // Set GL globals and forward the color:
    gl_Position = u_MVPMatrix * vec4(a_position_size.xyz, 1.0);
    gl_PointSize = a_position_size.w * pointScale;
    v_color = a_color;
}
```

Demo

- ▶ Particle system in WebGL:
 - ▶ <http://nullprogram.com/webgl-particles/>



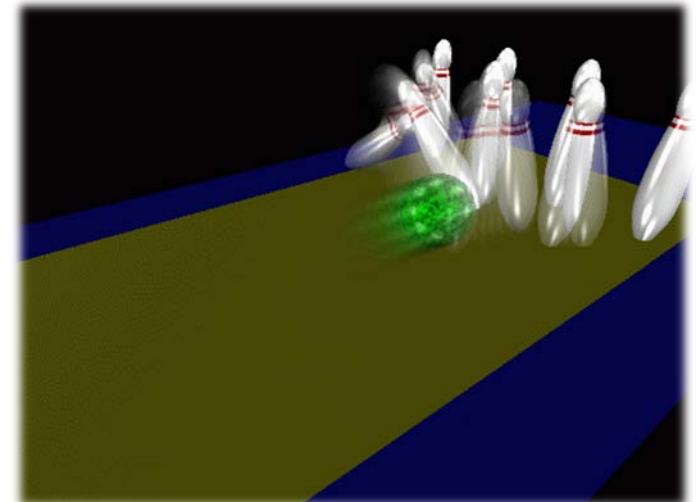
References

- ▶ Tutorial with source code by Bartłomiej Filipek, 2014:
 - ▶ <http://www.codeproject.com/Articles/795065/Flexible-particle-system-OpenGL-Renderer>
- ▶ Articles with source code:
 - ▶ Jeff Lander: “The Ocean Spray in Your Face”, Game Developer, July 1998
 - ▶ <http://www.darwin3d.com/gamedev/articles/col0798.pdf>
 - ▶ John Van Der Burg: “Building an Advanced Particle System”, Gamasutra, June 2000
 - ▶ http://www.gamasutra.com/view/feature/3157/building_an_advanced_particle_.php
- ▶ Founding scientific paper:
 - ▶ Reeves: “Particle Systems - A Technique for Modeling a Class of Fuzzy Objects”, ACM Transactions on Graphics (TOG) Volume 2 Issue 2, April 1983
 - ▶ <https://www.evl.uic.edu/aej/527/papers/Reeves1983.pdf>

Collison Detection

Collision Detection

- ▶ **Goals:**
 - ▶ Physically correct simulation of collision of objects
 - ▶ Not covered here
 - ▶ Determine if two objects intersect
- ▶ **Slow calculation because of exponential growth $O(n^2)$:**
 - ▶ # collision tests = $n*(n-1)/2$



Intersection Testing

- ▶ **Purpose:**
 - ▶ Keep moving objects on the ground
 - ▶ Keep moving objects from going through walls, each other, etc.
- ▶ **Goal:**
 - ▶ Believable system, does not have to be physically correct
- ▶ **Priority:**
 - ▶ Computationally inexpensive
- ▶ **Typical approach:**
 - ▶ Spatial partitioning
 - ▶ Object simplified for collision detection by one or a few
 - ▶ Points
 - ▶ Spheres
 - ▶ Axis aligned bounding box (AABB)
 - ▶ Pairwise checks between points/spheres/AABBs and static geometry

Sweep and Prune Algorithm

- ▶ Sorts bounding boxes
- ▶ Not intuitively obvious how to sort bounding boxes in 3-space
- ▶ Dimension reduction approach:
 - ▶ Project each 3-dimensional bounding box onto the x,y and z axes
 - ▶ Find overlaps in 1D: a pair of bounding boxes can overlap if and only if their intervals overlap in all three dimensions
 - ▶ Construct 3 lists, one for each dimension
 - ▶ Each list contains start/end point of intervals corresponding to that dimension
 - ▶ By sorting these lists, we can determine which intervals overlap
 - ▶ Reduce sorting time by keeping sorted lists from previous frame, changing only the interval endpoints

