Computer Graphics

- Advanced Rasterization -

Stefan Lemme
Recap: occlusion query

- Occlusion queries: simplified Ray-Tracing operations

- Normal ray-scene intersection:
  find **first** intersection with scene

- Occlusion-query:
  find **any** intersection with scene (slightly faster)

- Rasterization context: ray-scene intersection operation is not available
Shadow Techniques

• **Projective Shadows (on plane)**
  – Project all vertices onto (offset) receiver plane
  – Draw black triangles with (e.g. 50%) transparency
  – Must avoid multiple overdraw (“double blending”)
    • Draw receiver with unique stencil value
    • Draw shadows only stencil is set
    • Unset stencil while drawing shadows

• **Shadow Volumes**
  1. Draw scene without lighting
  2. Set stencil to 0 (1 if camera is inside volume)
  3. Turn off writing to depth and color buffers
  4. Draw volume, culling back faces, incrementing stencil buffer
  5. Draw volume, culling front faces, decrementing stencil buffer
  6. Draw scene with direct lighting, but only where stencil == 0
  7. Repeat from 2 for every light source
Shadow Volumes
Shadow Maps

- **Problem of Shadow Volumes**
  - Can have huge overdraw for complex objects – expensive
    - Especially when polygons span the view frustum

- **Idea:**
  - Render the scene from the viewpoint of the light, storing depth
  - At each pixel, transform the visible point into view from the light
    - Computing pixel and depth in that view (simple matrix transform)
    - Compare depth to the depth value, stored in the light map
    - If map depth is smaller, than the point is in shadow – skip
      - Otherwise do normal shading and add color to frame buffer
  - Repeat for every light source
Shadow Mapping

Light Source

Shadow Map

Camera
Shadow Maps: Principal Problems

- **Sampling**
  - Shadow maps are discretely and regularly sampled (e.g. grid)
  - Surfaces can have arbitrary orientation with respect to light
    - Can result in very bad sampling of a surface
  - Essentially impossible to solve
    - Would need adaptive sampling
    - But the shadow map has to be generated in advance, no feedback
    - Solved in ray tracing, as we generate the sample adaptively

- **Resolution**
  - Objects far from the camera should not be sampled finely
    - But shadow maps use a fixed grid
  - Must adapt to preferred resolution
    - Use several resolutions
      - E.g. Split or Cascaded Shadow Maps
    - Transform geometry appropriately
      - E.g. Perspective or Trapezoid Shadow Maps
Shadow Maps: Principal Problems

- **Interpolation/Filtering**
  - Shadow maps contain point samples
    - We know nothing about what happens in between
    - Regular leads to self-occlusion (in red)
  - Essentially impossible to solve without area information
    - E.g. min/max on depth
  - Approaches (selected)
    - Polygon offset
      - Simply shift the depth values by some value
      - Do so proportional to cos of angle
    - Percentage Closer Filtering:
      - In SW: Randomly sample pixel footprint and compute ratio
      - In HW: bi-linearly interpolate depth difference from neighboring pixels
    - Variance Shadow Maps:
      - Store higher order information for better interpolation
Shadow Map Filtering

- **Percentage-Closer Filtering**
  - Map area representing pixel to texture space
  - Stochastically sample pixel to find percentage of surface in light

![Shadow Map Diagram]
Percentage-Closer Filtering
Some Shadow Map Algorithms :-) 

- **Simple**
  - SSM "Simple"  
- **Splitting**
- **Warping**
- **Smoothing**
- **Filtering**
- **Soft Shadows**
- **Assorted**
  - CSSM "Camera Space" [http://free-zg.t-com.hr/cssm/](http://free-zg.t-com.hr/cssm/)
  - DASM "Deep Adaptive"
  - DPSM "Dual Paraboloid" [http://sites.google.com/site/osmanbrian2/dpsm.pdf](http://sites.google.com/site/osmanbrian2/dpsm.pdf)
  - LPSM "Logarithmic" [http://gamma.cs.unc.edu/LOGSM/](http://gamma.cs.unc.edu/LOGSM/)
Ambient Occlusion

- Calculates shadows against assumed constant ambient illumination
  - Idea: in most environments, multiple light bounces lead to a very smooth component in the overall illumination
  - For this component, incident light on a point is proportional to the part of the environment (opening angle) visible from the point
  - Describes well contact shadows, dark corners

\[ \text{assume constant light outside radius } r \]
Ambient Occlusion (Visibility)
AO Using Ray-Tracing

- Computation using Ray-Tracing straight forward
  - Start at point P
  - Sample N directions (D₁-Dₙ) from upper hemisphere
  - Shot shadow rays from P to Dᵢ with maximum length r
  - Count how many rays reach the environment
  - Gives correct result in the limit, but requires many rays to avoid noise (i.e. very slow)

![Diagram showing ray tracing](image)
AO Using Ray-Tracing
Screen Space Ambient Occlusion

- Can we approximate ambient occlusion in real-time?
- Ray-scene intersection too slow

- **Idea: use z-buffer as scene approximation**
  - Horizontal and vertical position give position of point in x,y-direction (camera space)
  - Z-buffer content gives position of point in z-direction (camera space)
  - Contains discrete representation of all *visible* geometry
  - Use ray-tracing against this simplified scene
Screen Space Ambient Occlusion

camera

z-buffer

fake corner?

corner

geometry outside viewport?

viewport?
Screen Space AO

• **Tracing many rays is still expensive**
  – Often 200 and more samples are needed for good results

• **Approach**
  – For each pixel (Crytek approach, many others available)
    • Test a number of random points in sphere visible 3D point
      – Do not know surface orientation, so must test in all directions
    • If more than 50% pass we have full visibility
      – Otherwise scale AO with number of samples
    • Can still be quite costly
  – **Acceleration**
    • Use different pseudo-random pattern for each pixel in NxN block
      – Gives slightly different values for each pixel
    • Filter over a NxN neighborhood
      – Uses all samples: E.g. 4x4 block with 16 samples each: 256 samples total
    • Make sure not to filter over wrong pixels (background)
      – Take distance, normal, etc. into account (→ bilateral filter)
Screen Space AO

Crytek
Screen Space AO

Crytek
Deferred Shading

- Screen-space shading technique
- Avoid over-shading of fragments due to later occlusion

- First pass gathers data relevant to shading into G-Buffer
  - Color (albedo)
  - Normal
  - Depth

- Second pass performs actual shading per pixel (i.e. only for visible fragments)

https://de.wikipedia.org/wiki/Datei:Deferred_Shading_FBOs.jpg
Volume Rendering

- Texture-based volume rendering using view-aligned slicing of volume data
- Proxy-Geometry for rasterization
- Draw in back-to-front sorted order with alpha blending enabled
Isosurfaces from Volume Data
Isosurfaces from Volume Data

- originated by William E. Lorensen and Harvey E. Cline in 1987
- $\text{caseBit}[i] = \text{density}(v_i) > 0$

\[
\text{case} = v_7|v_6|v_5|v_4|v_3|v_2|v_1|v_0
\]
\[
= 11000001
\]
\[
= 0xC1 = 193
\]
Isosurfaces from Volume Data

- 15 fundamental cases for Marching Cubes
Isosurfaces from Noise
Procedural Terrain Generation
Procedural Terrain Generation
Decorating large-scale Terrain
Decorating large-scale Terrain

- goal: cover large terrain surfaces with grass in real-time
- thousands of millions of grass blades
- multiple instances of a single grass patch – three different representations
- arranged into the cells of a uniform grid
Level of Detail

- **image-based grass**
- **geometry-based grass**
- **volume-based grass**
- **only geometry-based representation**
- **transition zone with multiple representations in use**
- **only more abstract representations**

Distance to camera
Decorating large-scale Terrain
Rendering Text and Decals
Rendering Text and Decals

- Font Rendering
- Glyph consists of splines as outline
Rendering Text and Decals

- Bitmap Fonts
Rendering Text and Decals

- Magnification using semi-transparent textures

Valve

64x64 texture, alpha-blended

64x64 texture, alpha tested
Rendering Text and Decals

- Magnification using distance fields

High resolution input

64x64 Distance field

Valve
Rendering Text and Decals

- Magnification using distance fields

Valve

- 64x64 texture, alpha-blended
- 64x64 texture, alpha tested
- 64x64 texture, distance field
Seminar “Real-Time Rendering”

• **Summer Term 2019**
  – Focus on rasterization and inner workings of graphics APIs
  – Implement a rendering technique using a software rasterizer

• **Modus operandi**
  – Each student works solely on his own topic
  – Individual supervision by a CG member
  – Mid-term short presentation
  – End-term presentation incl. implementation and live demo
  – Documentation of work in the fashion of a short paper
Seminar “Real-Time Rendering”

• Non-exhaustive list of topics may include …
  – Procedural Content
    (fractals, wavelets, procedural materials, procedural geometry …)
  – Deferred Rendering
    (G-Buffers, deferred shading, deferred lighting, HDR, …)
  – Culling
    (view-frustum culling, occlusion culling, hierarchical depth culling,
    portals and visibility pre-computation, …)
  – Processing Geometry
    (splines, surface subdivision, simplification, geometry and
tesselation shaders, …)
  – Compressed Images
    (textures, framebuffers, fast decompression, GPU-friendly
storage, color and normal encoding, …)
  – …

• More Info on the website soon:
  https://graphics.cg.uni-saarland.de/courses/