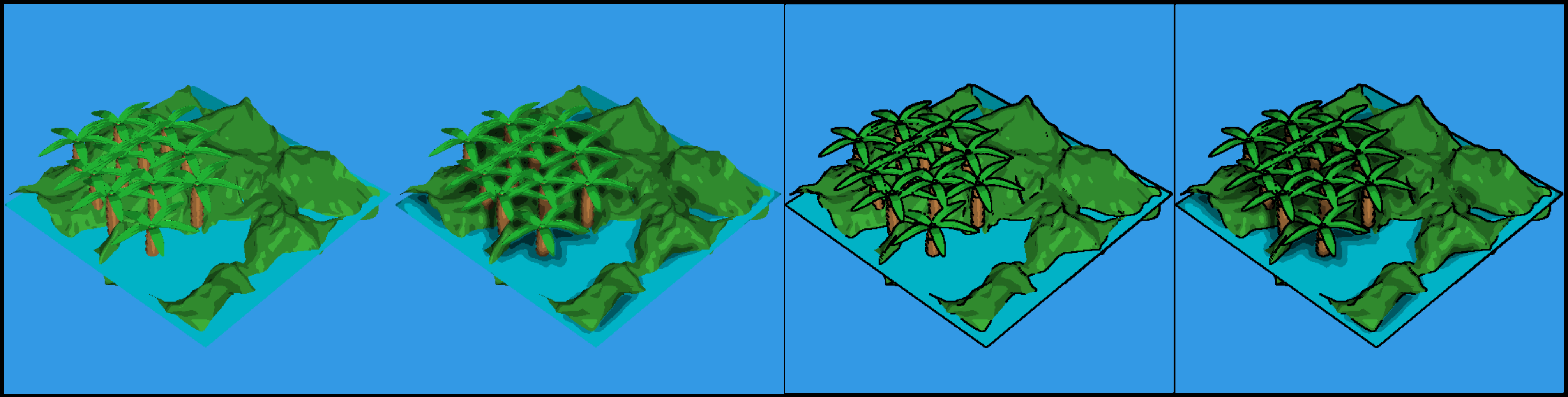
Combining Screen-Space Ambient Occlusion and Cartoon Rendering on Graphics Hardware



Dan Nottingham and Brett Lajzer

Overview

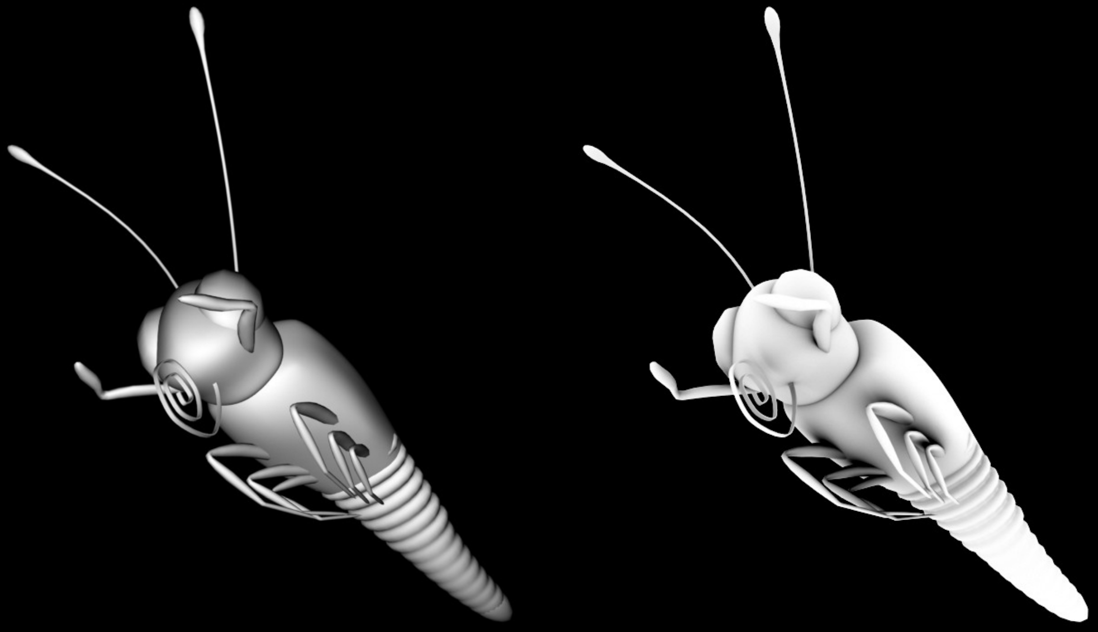
- Motivation
- Ambient Occlusion and SSAO
- Cartoon Rendering
- Hardware Implementation
- Demo
- Future work

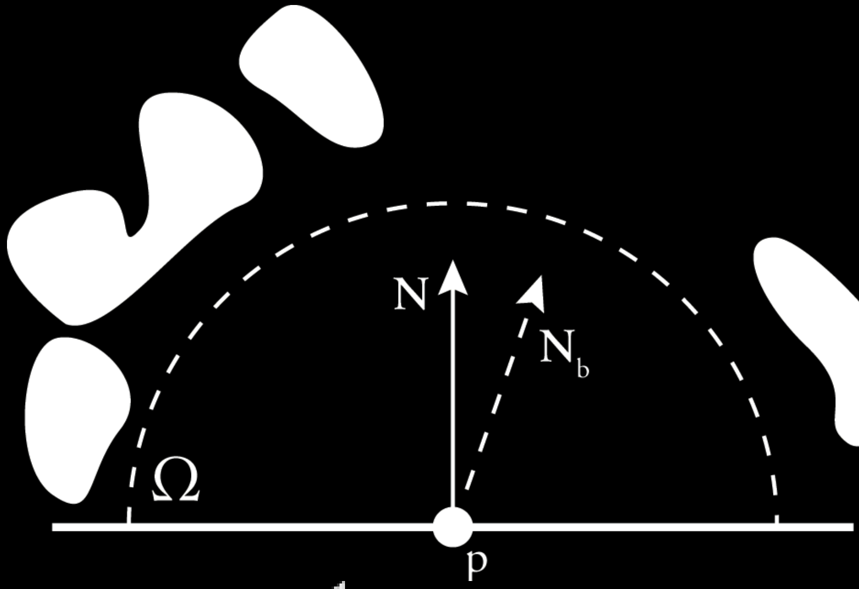
Motivation

- NPR techniques like cel shading look cool
- But, details are often lost, and scene looks flat
- SSAO helps bring out features, especially creases and object boundaries
- Hardware implementation allows for real-time/interactive frame rates for dynamic scenes, such as in games

Ambient Occlusion

- Approximation to global illumination
- Point on a surface receives less ambient light if there are occluding objects nearby in its hemisphere
- Usually calculated with Monte-Carlo ray casting





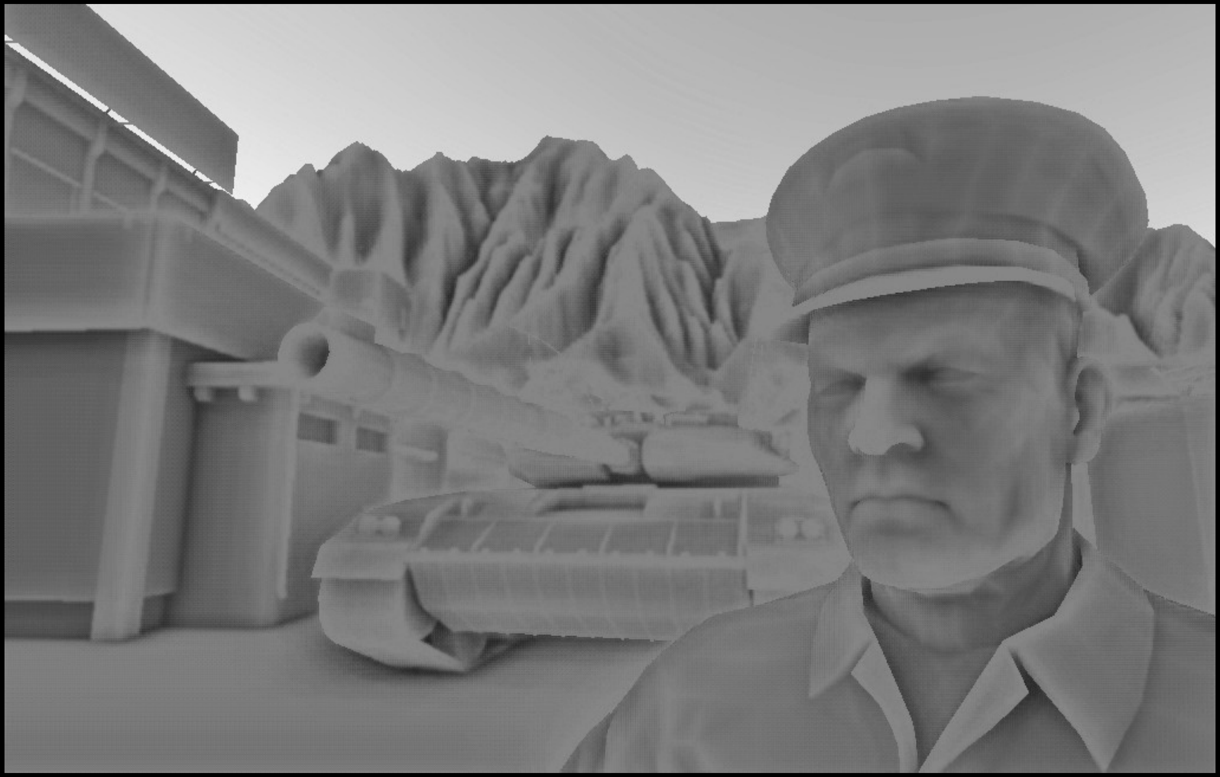
$$A_p = \frac{1}{\pi} \int_\Omega V_{p,\omega}(N \cdot \omega)\, d\omega$$

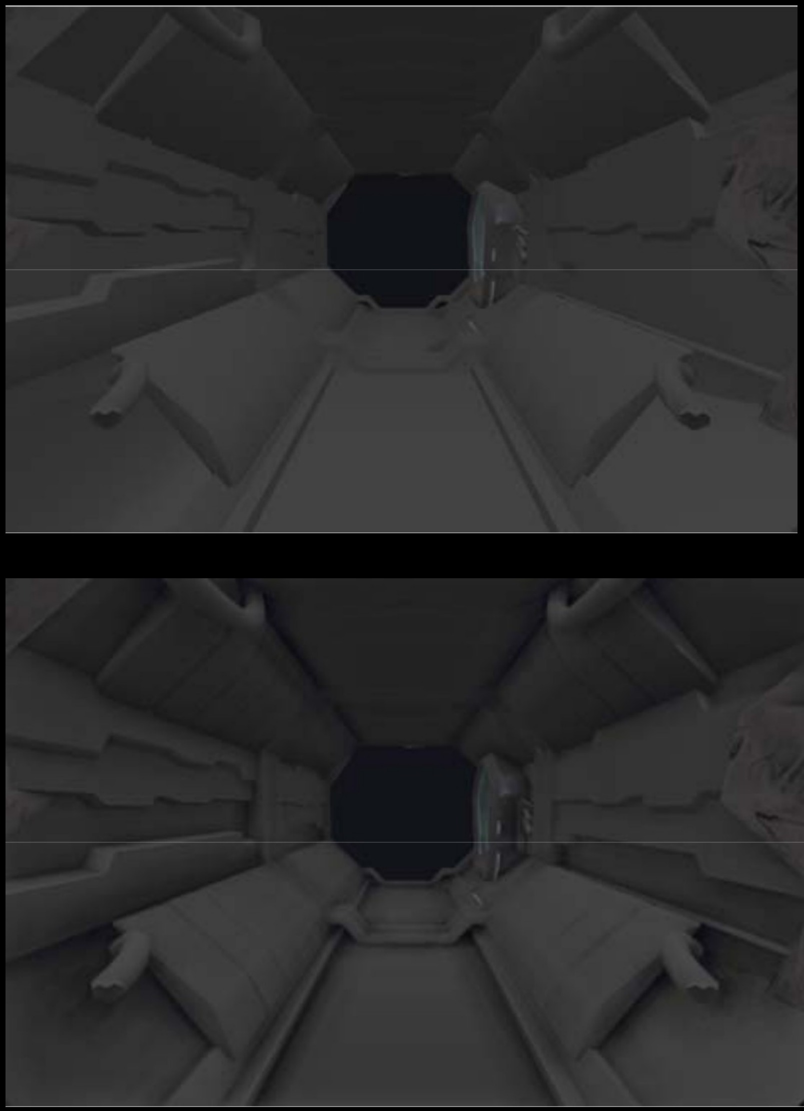
http://en.wikipedia.org/wiki/Ambient_occlusion

Screen-Space Ambient Occlusion

- Approximate AO using the depth buffer
- For target pixel, determine occlusion by:
  - Taking random sample points in hemisphere
  - Comparing depth of sample point to depth buffer
  - If sample is behind stored depth and not too far behind, point is occluded
- Can be implemented in hardware

Screen-Space Ambient Occlusion

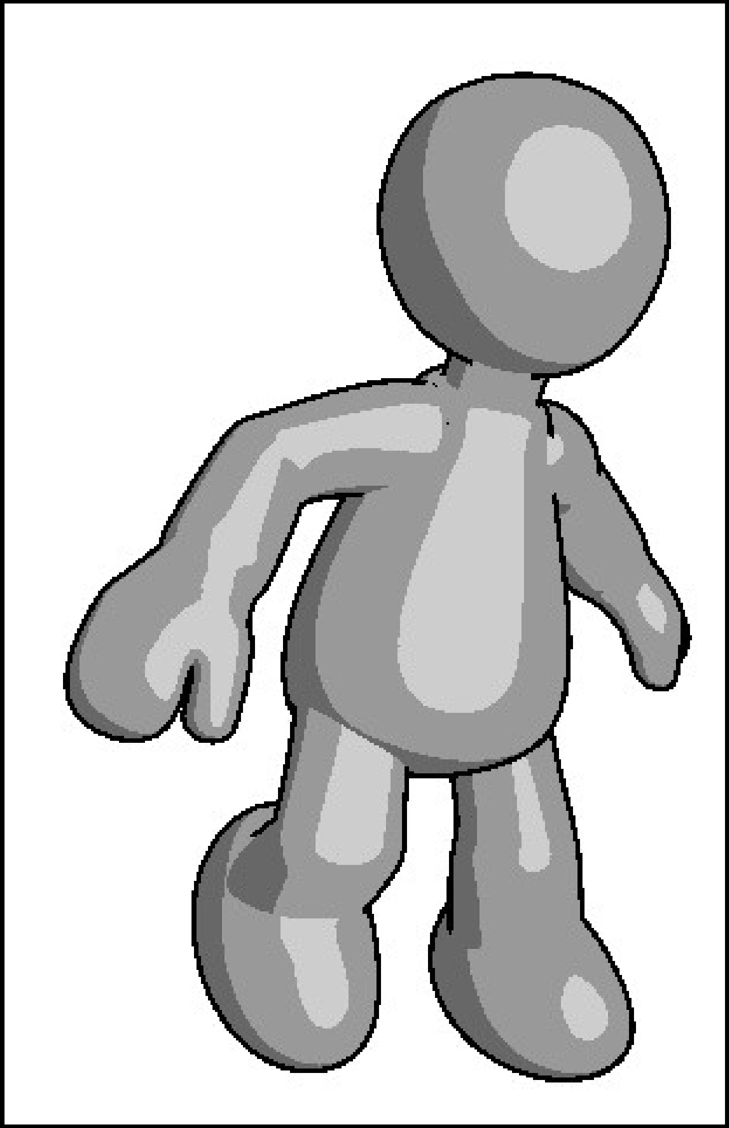




From “Finding Next Gen – CryEngine 2”
by Martin Mittring, Crytek GmbH

Cartoon Rendering

- Draw outlines based on discontinuities in depth and camera-space normals
- Threshold lighting intensities to discreet values



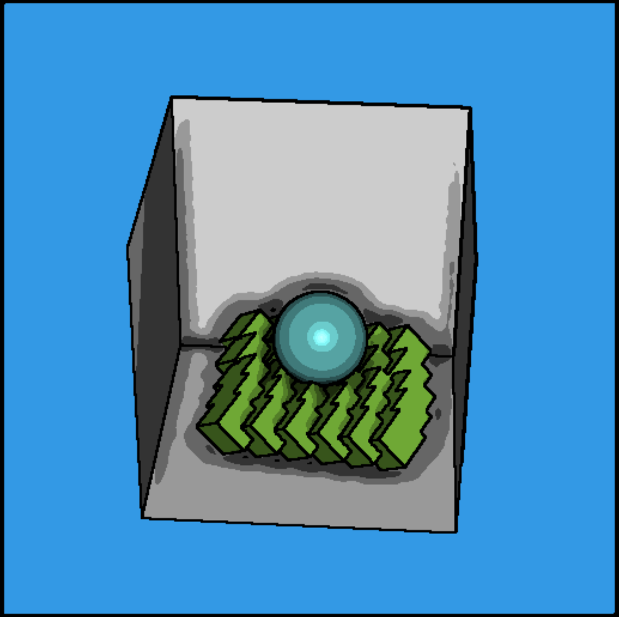
From “Non-Photorealistic Rendering with Pixel and Vertex Shaders”, Drew Card and Jason L. Mitchell, ATI Research

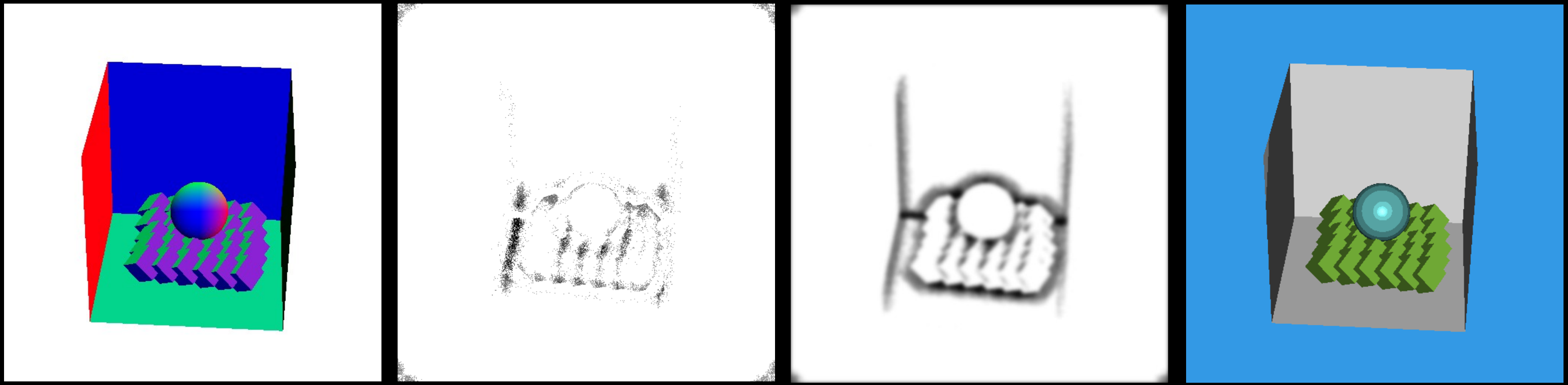
Hardware Implementation

- Used OpenGL and GLSL to implement shaders
- Most work done in fragment (pixel) shaders
- Intermediate results rendered to textures through frame buffer objects
- For certain steps, render a single quad filling the whole screen, textured with results of previous steps

Implementation – Rendering Passes

- Camera-space depth and normals
- SSAO
- Blur SSAO, repeat 10 times
- Lighting, color, and cel shading
- Outlines



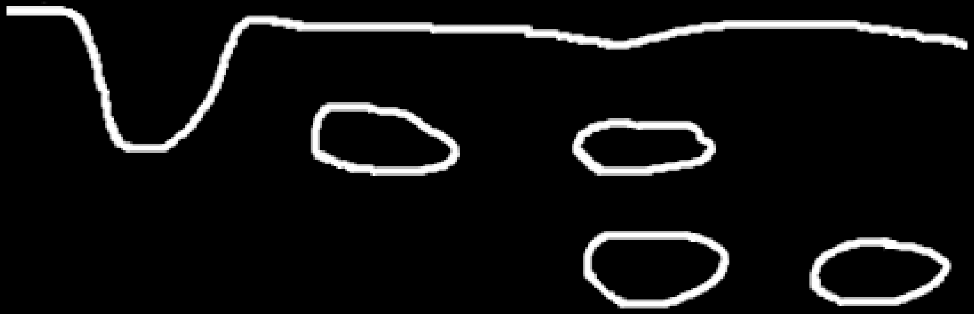


SSAO Implementation

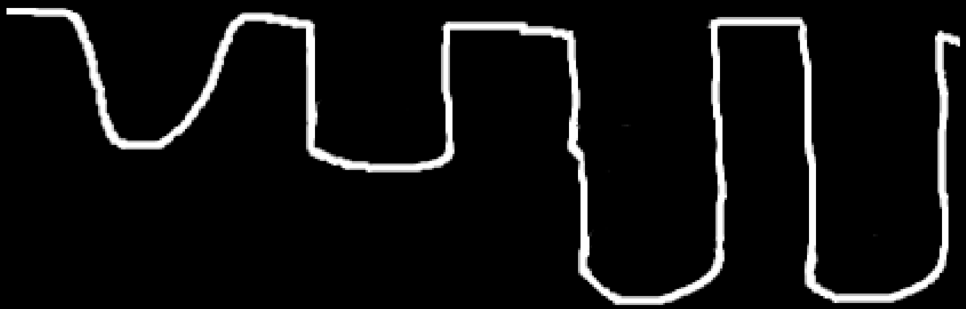
- Shader is given 8 sample points on a sphere and random vector texture
- Randomize sample points by reflecting around vector at pixel
- Increase occlusion value if pixel depth is behind depth buffer at sample
- Throw out occlusion values less than 5, map remainders from 0 to 1 quadratically

SSAO Implementation

- Loss of information due to depth buffer
  - Bump is equivalent to object offset from surface
- Crysis implementation does not occlude if depth difference is too large
  - Leads to under-occlusion in second-from-right case
- Our implementation ignores how far occluder is from surface in z
  - Causes over-occlusion in rightmost case

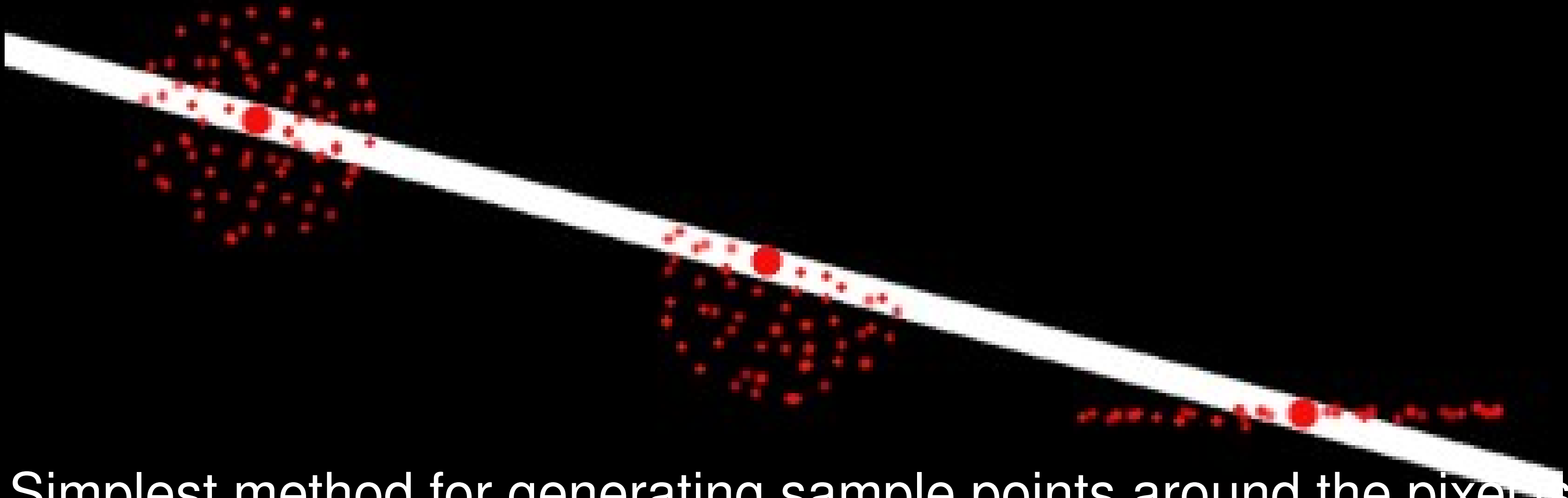


Scene from top-down view



Same as seen from depth-buffer

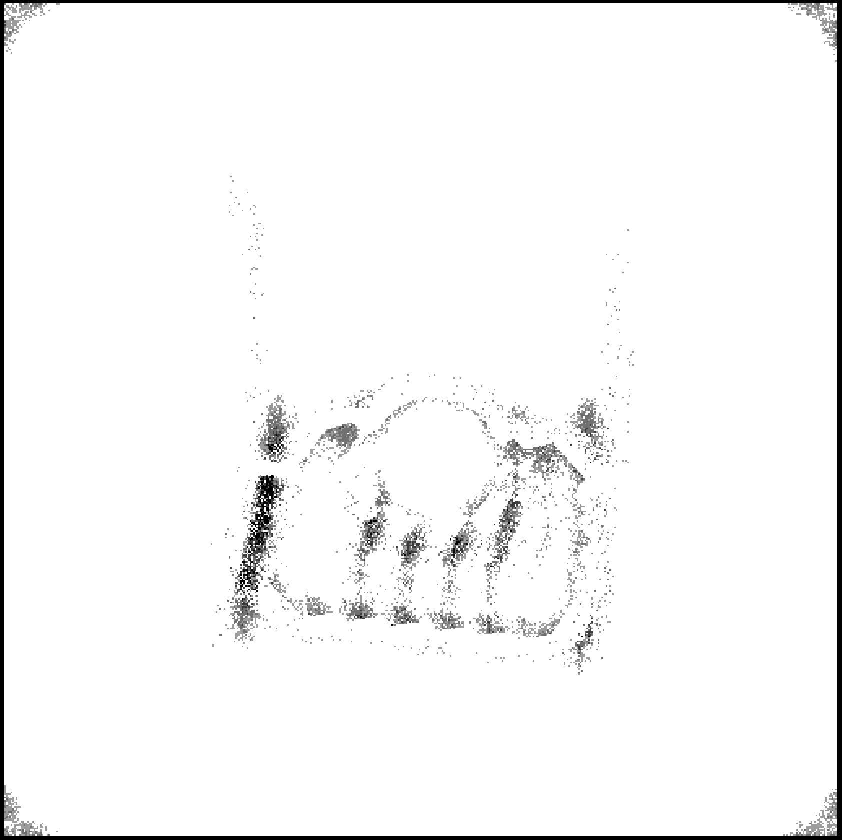
SSAO Implementation

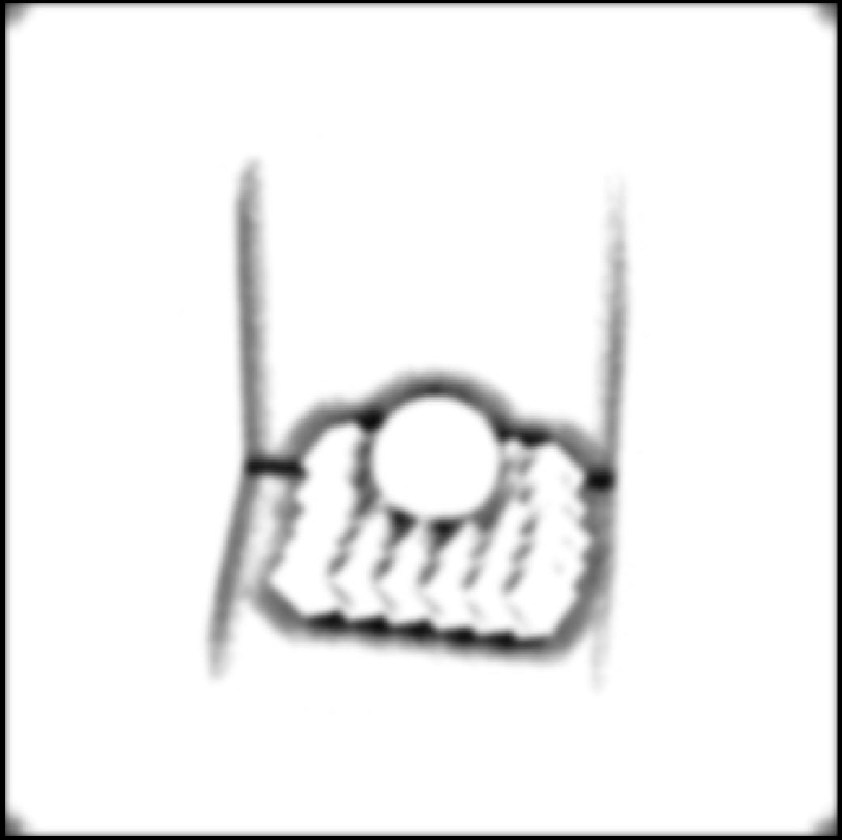


- Simplest method for generating sample points around the pixel is to take samples within the sphere centered at it (left)
- Sampling hemisphere in direction of normal is more correct, prevents self-occlusion (center)
- We generate points within a sphere, but essentially flatten the points on to a plane in how we do our comparison (right)
  - Our mapping mostly eliminates self-occlusion

SSAO Implementation

- Result is very noisy, so we repeatedly apply a 3x3 Gaussian blur filter



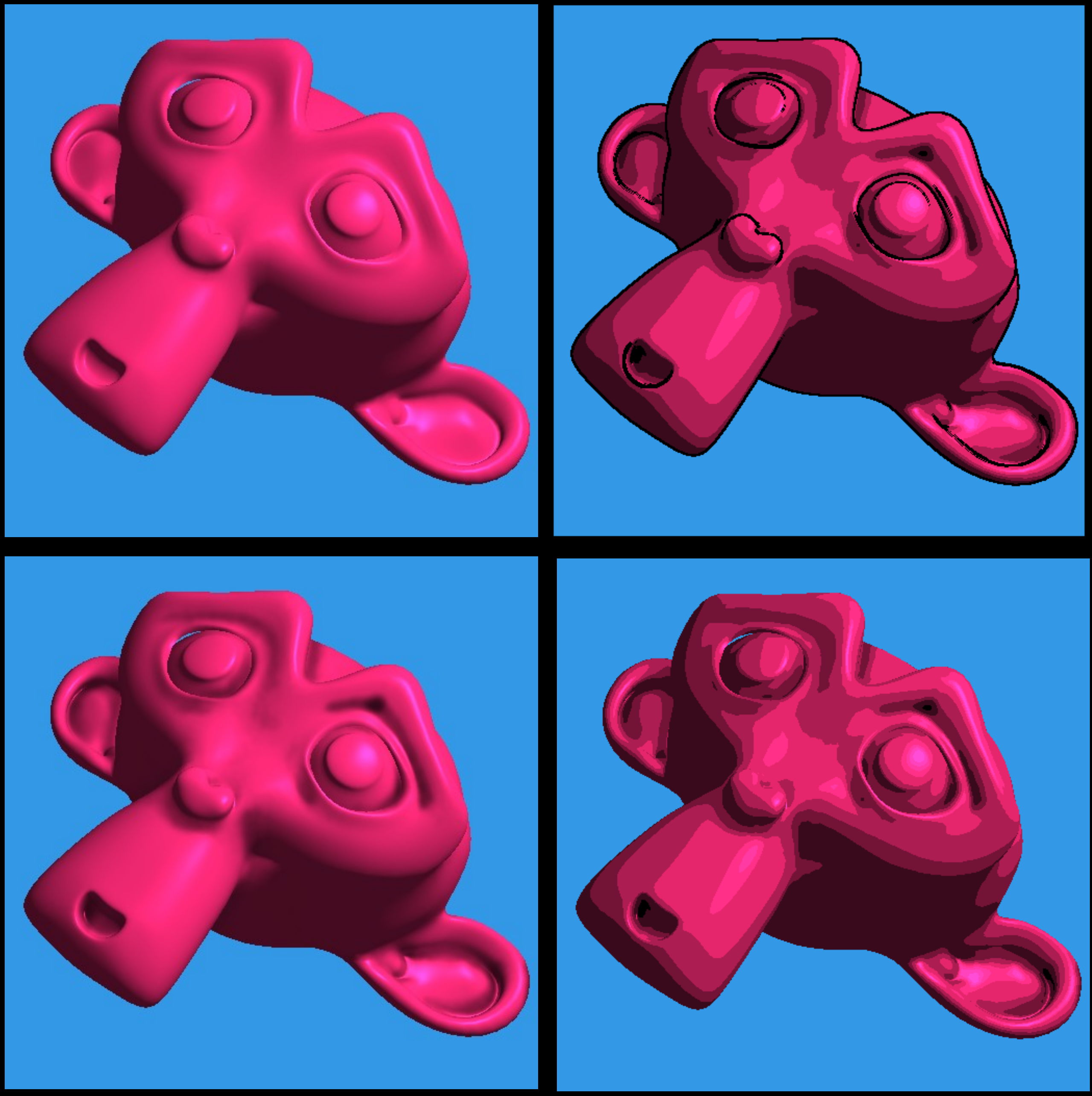


Cartoon Rendering Implementation

- We calculate diffuse and specular terms of Phong lighting, but determine intensity by:
  - ssao * (diffuse + ambient1) + specular + ambient2
- Lighting intensity discretized to five threshold values:
  - floor(intensity * 4.0) / 4.0
- Outlines are drawn on top in separate shader, by finding depth and normal discontinuities

Cartoon Rendering Implementation

- Counter-clockwise from top-left:
  - Phong only
  - Phong and AO
  - Threshold lighting for toon shading
  - Outlining added to toon shading

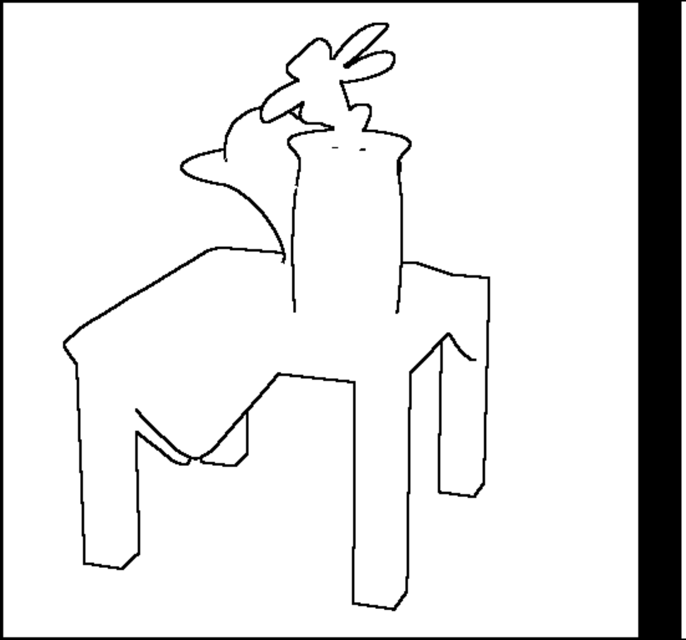


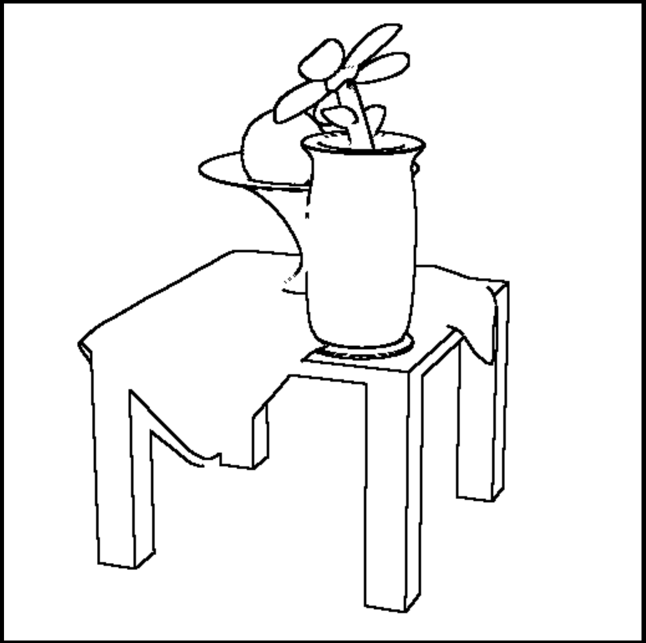
Cartoon Outlining

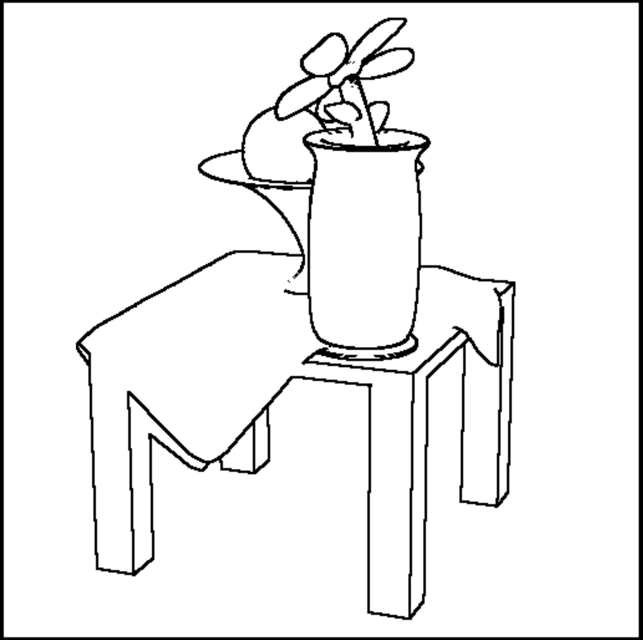
- Take 8 sample points in square around pixel
- Sample normals and depth at point and at pixel
- Depth discontinuity: difference in depth is greater than threshold
- Normal discontinuity: dot product between normals is less than threshold
- If there is any discontinuity, color pixel black

Cartoon Outlining

- Depth finds silhouettes, outer edges, fails when objects are too close
- Normals find sharp interior edges, plus outer edges in some cases







Demo

Future Work

- Better hardware (Pixel Shader 2.0 is limited to 768 instructions per shader, 3.0 allows up to 65536)
- Improvements to SSAO:
  - Eliminate objects self-occluding
  - Distance of occlusion not accounted for
  - Reduce noise, possibly through more sampling and/or better blurring (bilateral filtering?)
- Anti-aliased outlines, and better outline detection
- Apply other standard techniques: texture mapping, shadows
- Experiment with other NPR techniques

Division of Labor

- Brett
  - OpenGL rendering framework (nastiness)
  - SSAO shader, other misc. shaders
  - Test scene modeling
  - Pretty pictures for paper
- Dan
  - Mesh loader
  - Outlining shader and tweaks to SSAO and lighting shader
  - Most of paper write up and presentation