Advanced Computer Graphics
-
Hardware

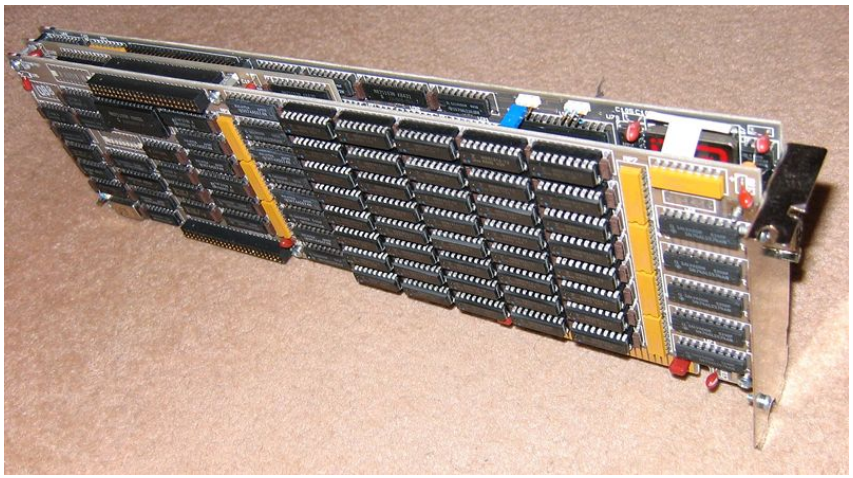
Matthew van der Zwan

RuG

15 maart 2010

Graphics Card

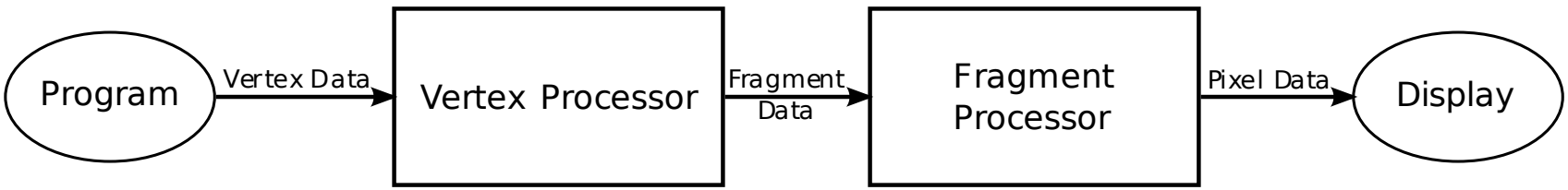
- First 3D capable card released in 1984
- Available to consumers in mid-1990's
- Programming API independent of hardware
  - OpenGL (begin 1990's)
  - DirectX (late 1990's)



Pipeline

The process of rendering graphics can be described as:

- Program sends vertices to the GPU
  - For simple applications this is all
  - More complicated data is possible too
  - Think of material colors, normals, textures
- The GPU interpolates fragments between the vertices
  - Extra data gets interpolated too
  - Alpha blending happens at this level too

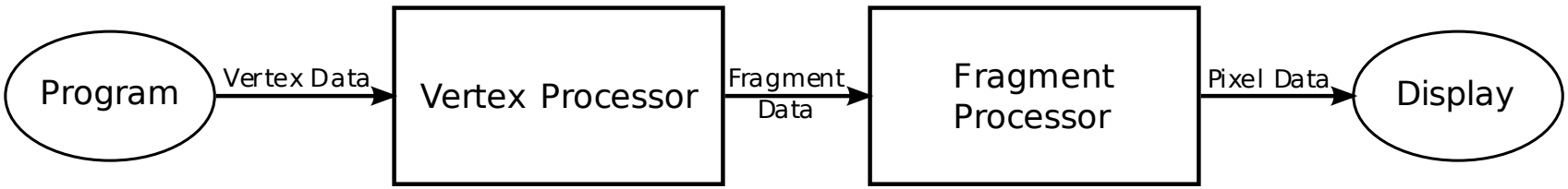


Pipeline

Limitations of the traditional pipeline:

- User is restricted to functionality provided by API:
  - Lighting
  - Texture effects

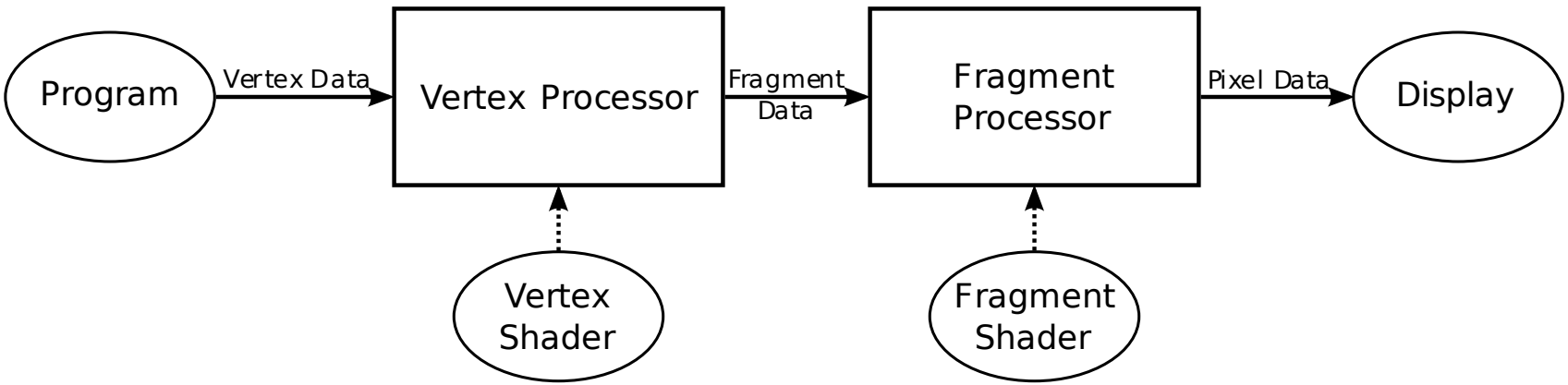
Solution: Shaders!



Shaders

Shaders:

- Provides the opportunity to go beyond standard features
- Gives control over processing of vertices
- Gives control over processing of fragments
- This allows (more) complete control



Shaders - What can we do?

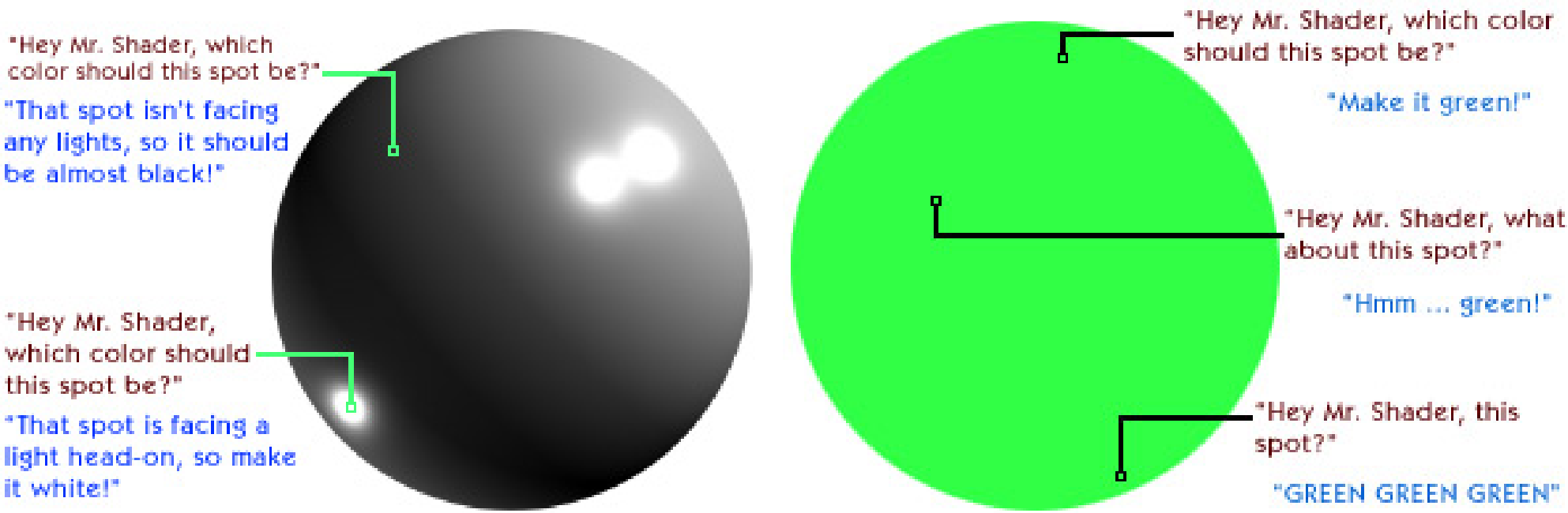
In the vertex shader we can:

- Move the vertex
- Determine/Change the normal of a point
- Not: add/delete a vertex

In the fragment shader we can:

- Set the color of a fragment
  - According to a texture
  - According to a lighting model
- Modify the depth value of a fragment

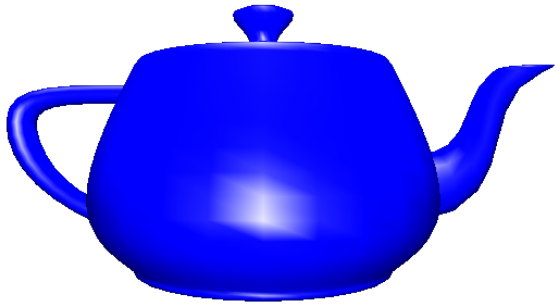
Shaders - Schematic working

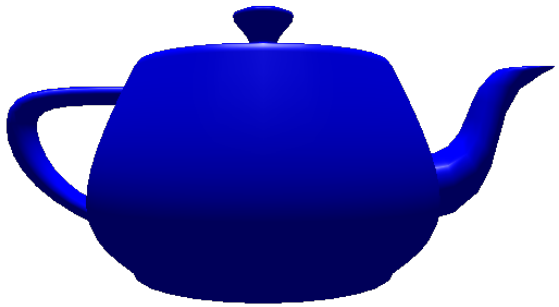


Applications of shaders

With shaders we can:

- Create realistic light effects
- Simulate natural phenomena
  - Fog, water, fire
- Use textures differently
  - Normal buffers, bump maps
- Implement reflection/refraction
- Perform more general computations
  - Image processing
  - Fluid simulations

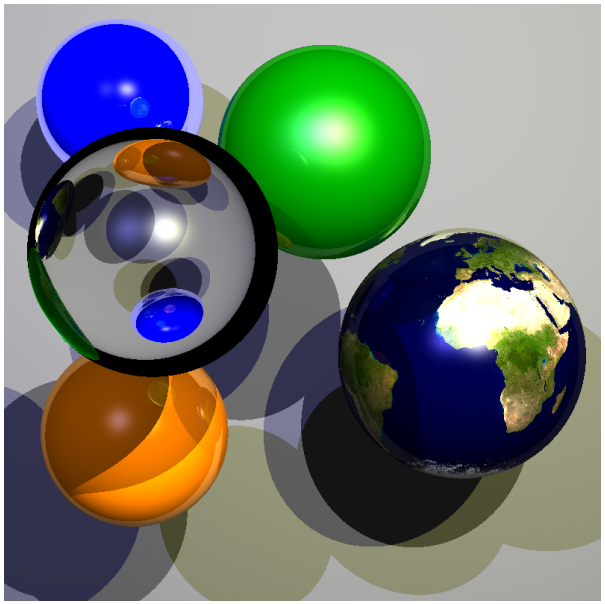




Texturing

Textures are images that can be applied to a primitive.
They can be used to:

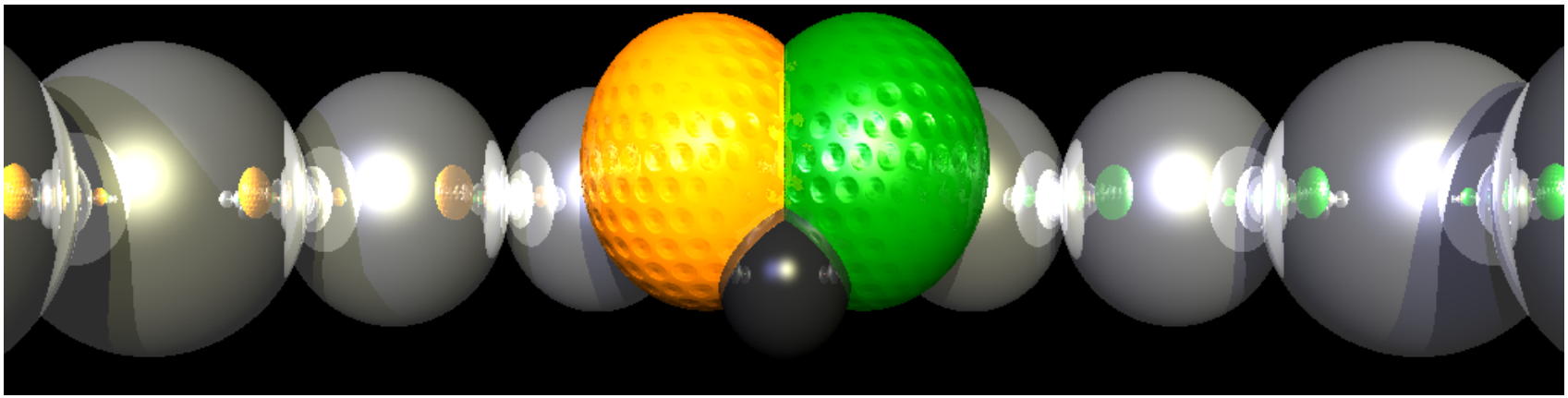
- Simulate a natural substance
  - Wood, marble
- Simulate a reflecting surface



Texturing

Textures can also be used as:

- Bump maps, when combined with shading
- A fast way to send arbitrary data to a shader



Commonly used texturing techniques:

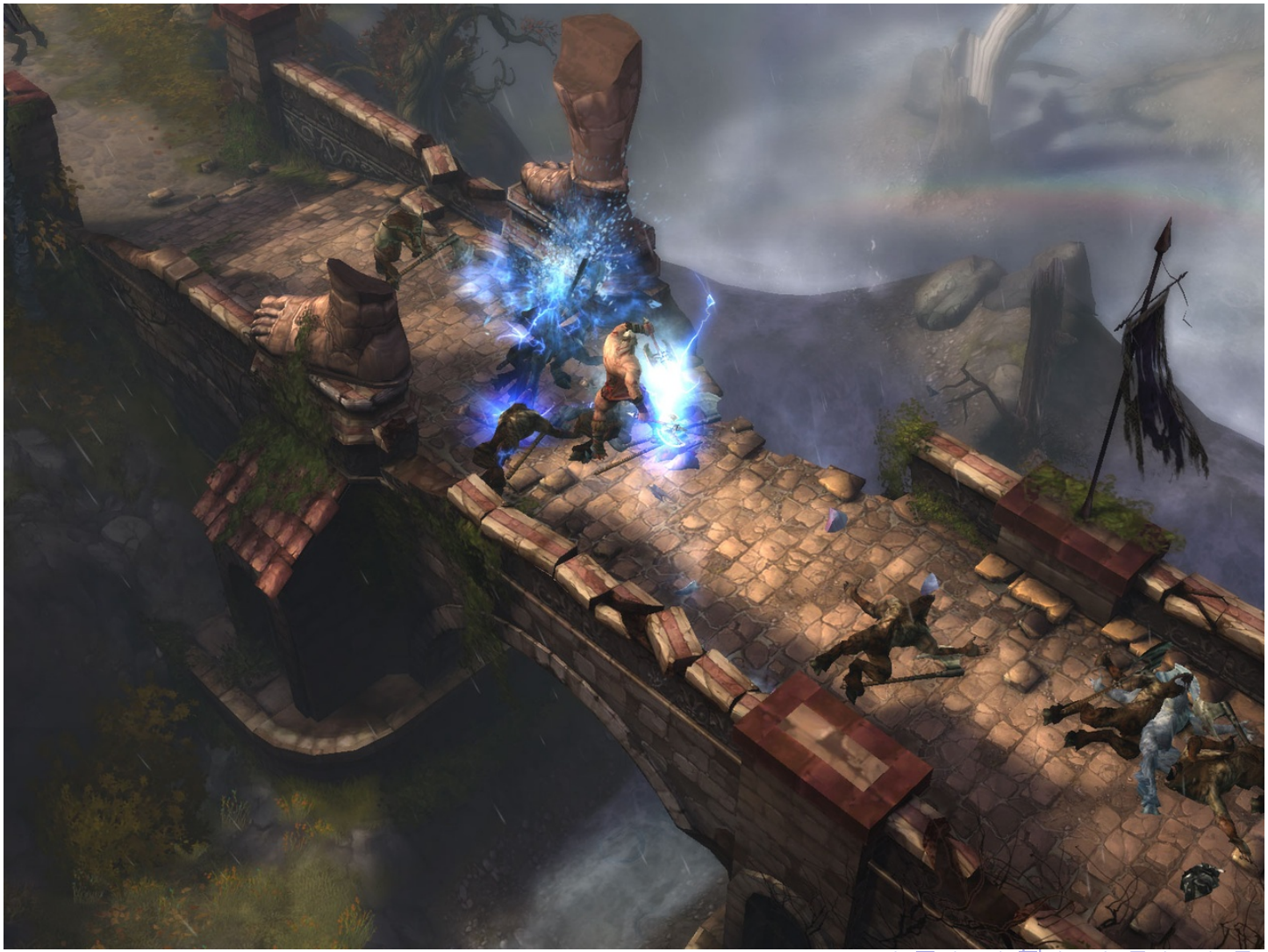
- Multi-texturing
- Mipmapping

Texturing - How it works

Work flow:

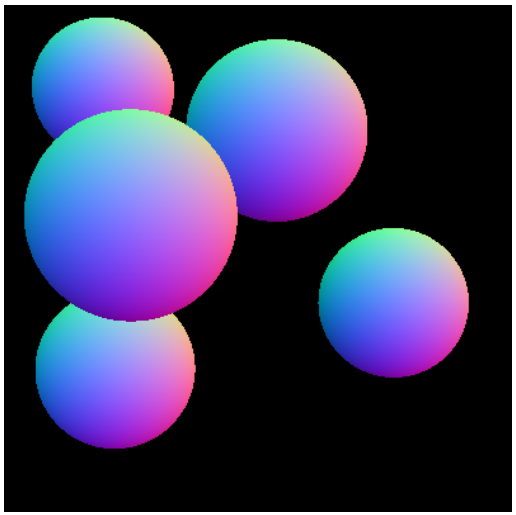
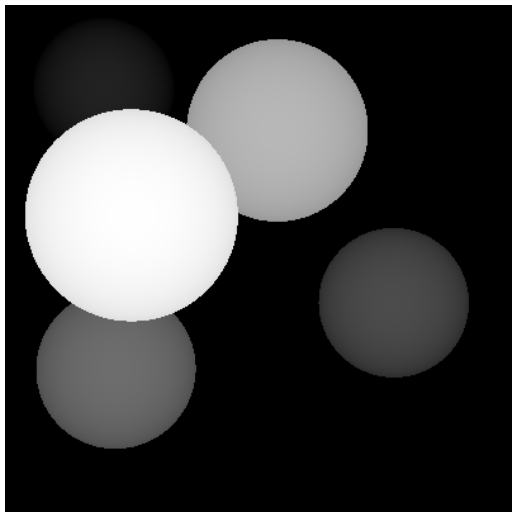
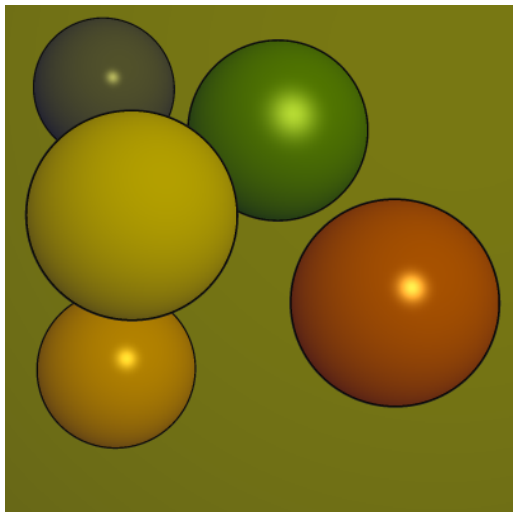
- Set necessary settings
  - Wrapping, magnification function
- For each vertex we define texture coordinates
- Texture coordinates are interpolated for fragments
- Texture elements are applied to fragments

Texturing - Result



Multi-Pass Rendering

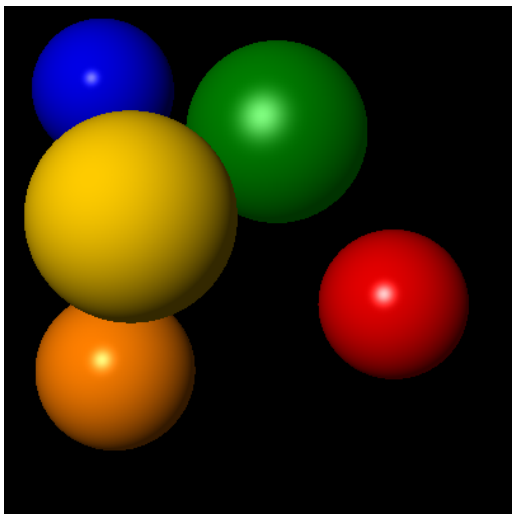
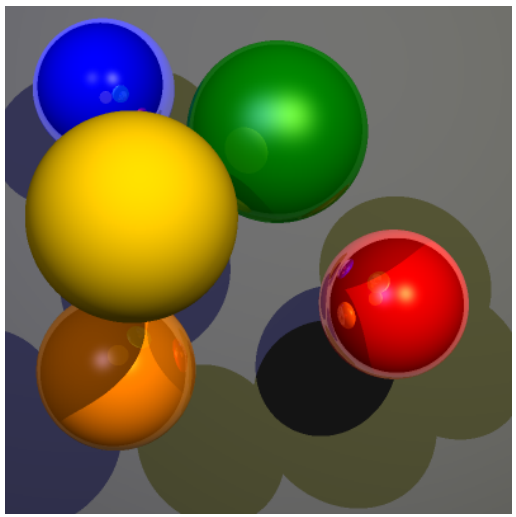
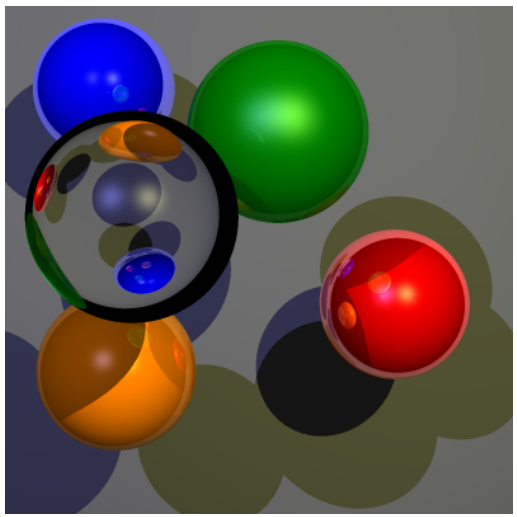
- Render the scene multiple times
- Each rendering pass
  - Change the view point
  - Use a different shader
- Save each pass to a buffer (texture)
- Combine the results

Multi-Pass Rendering

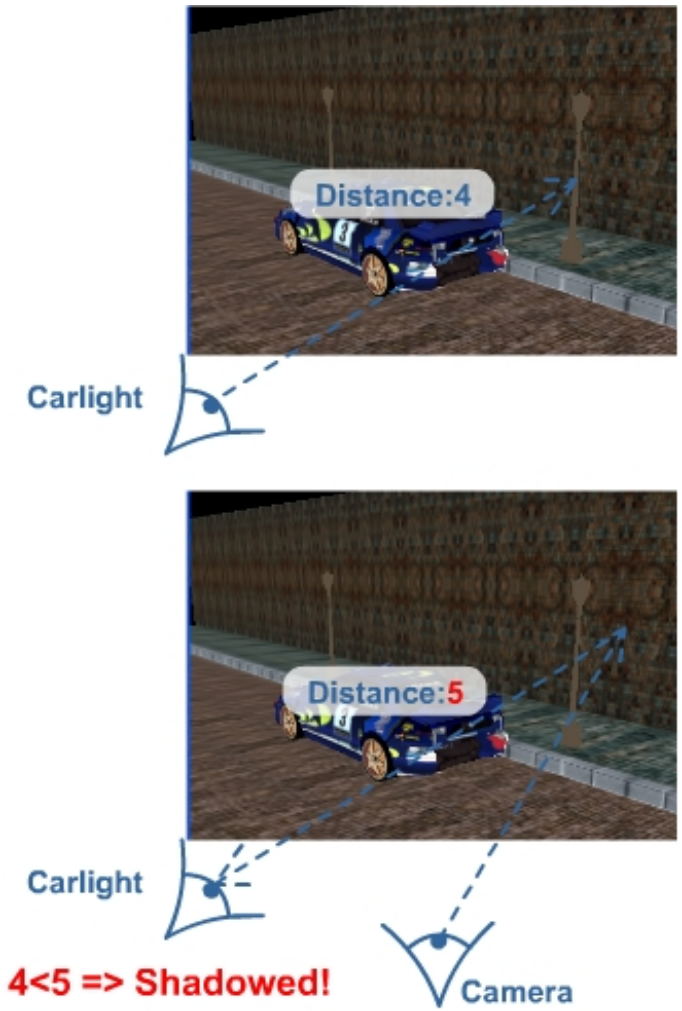
Example applications:

- Shadows
- Reflection
- Refraction

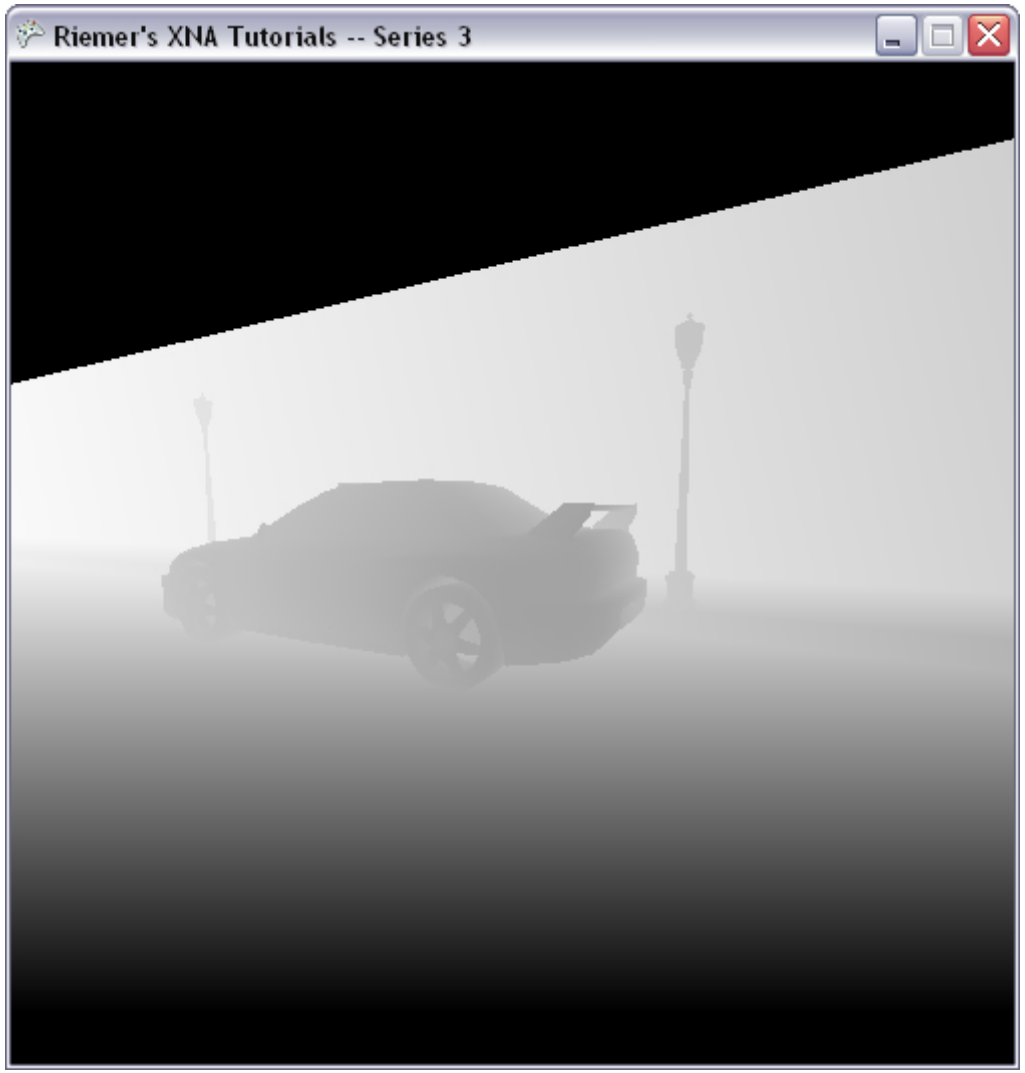




Multi-Pass Rendering - Shadows

Multi-Pass Rendering - Shadows



Multi-Pass Rendering - Shadows



Multi-Pass Rendering - Shadows

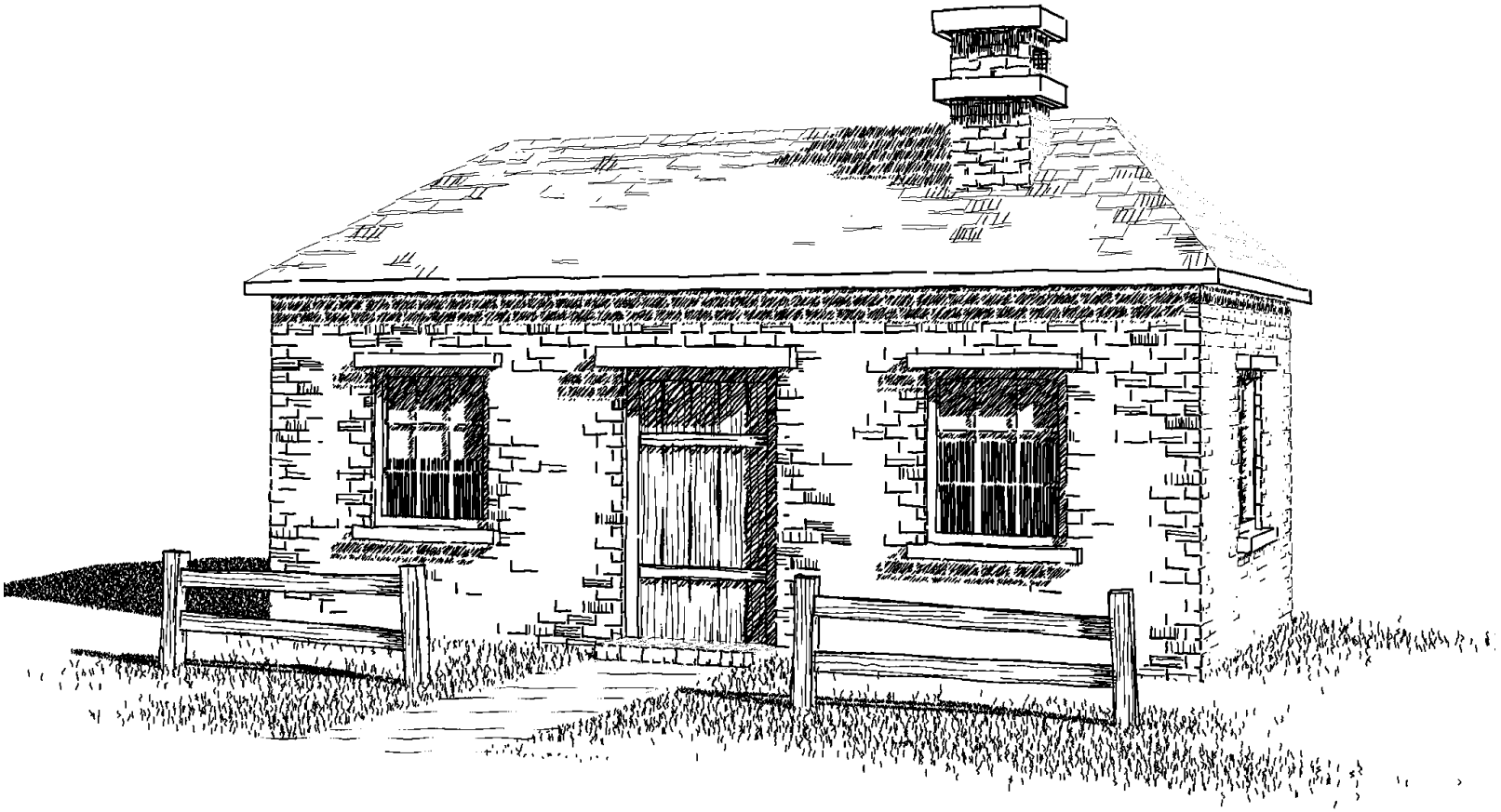
Multi-Pass Rendering - Shadows

Multi-Pass Rendering - Examples

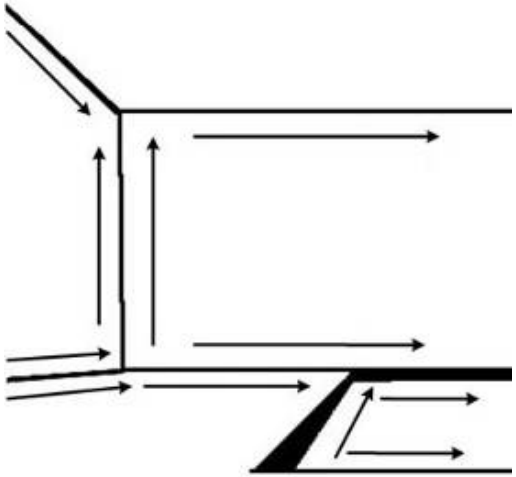
Example Videos

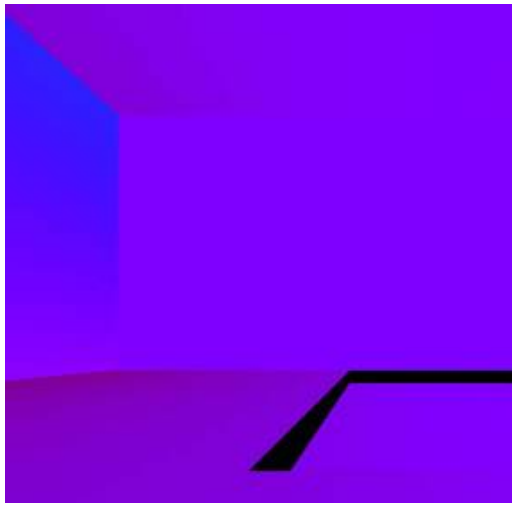
- Rendering to Textures
  - (http://www.youtube.com/watch?v=sTTj3bUA_Mg)
- More sophisticated effects
  - (http://www.youtube.com/watch?v=g_8sm5YZKiE)

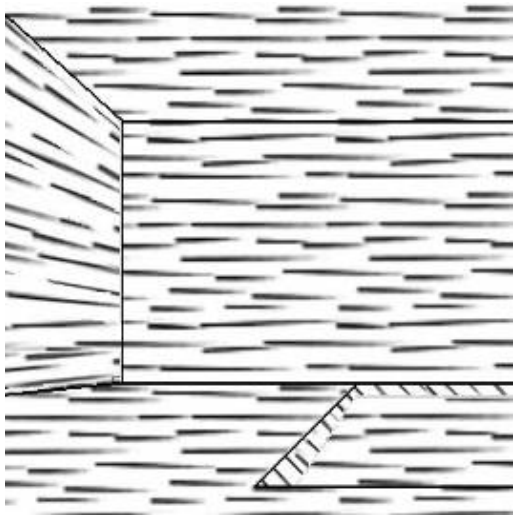
Example: Hardware hatching



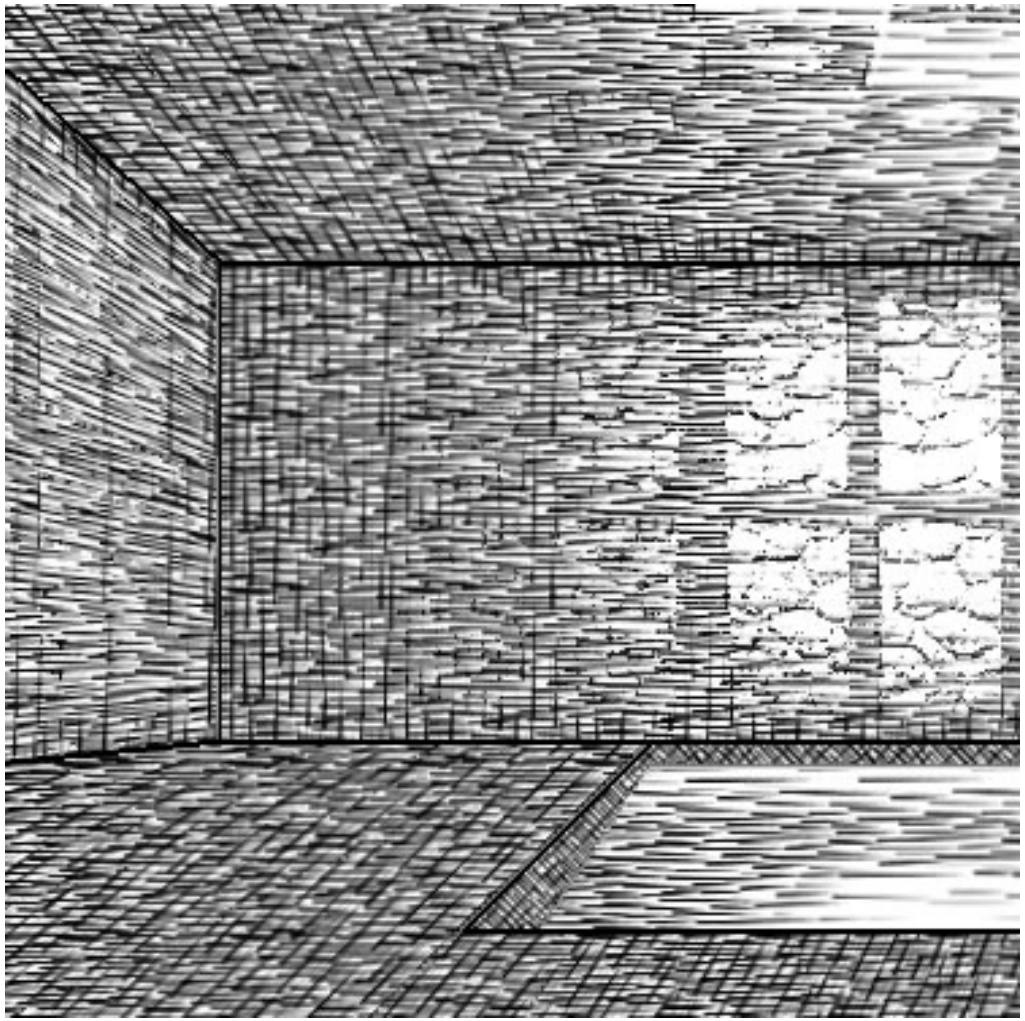
Direction of strokes



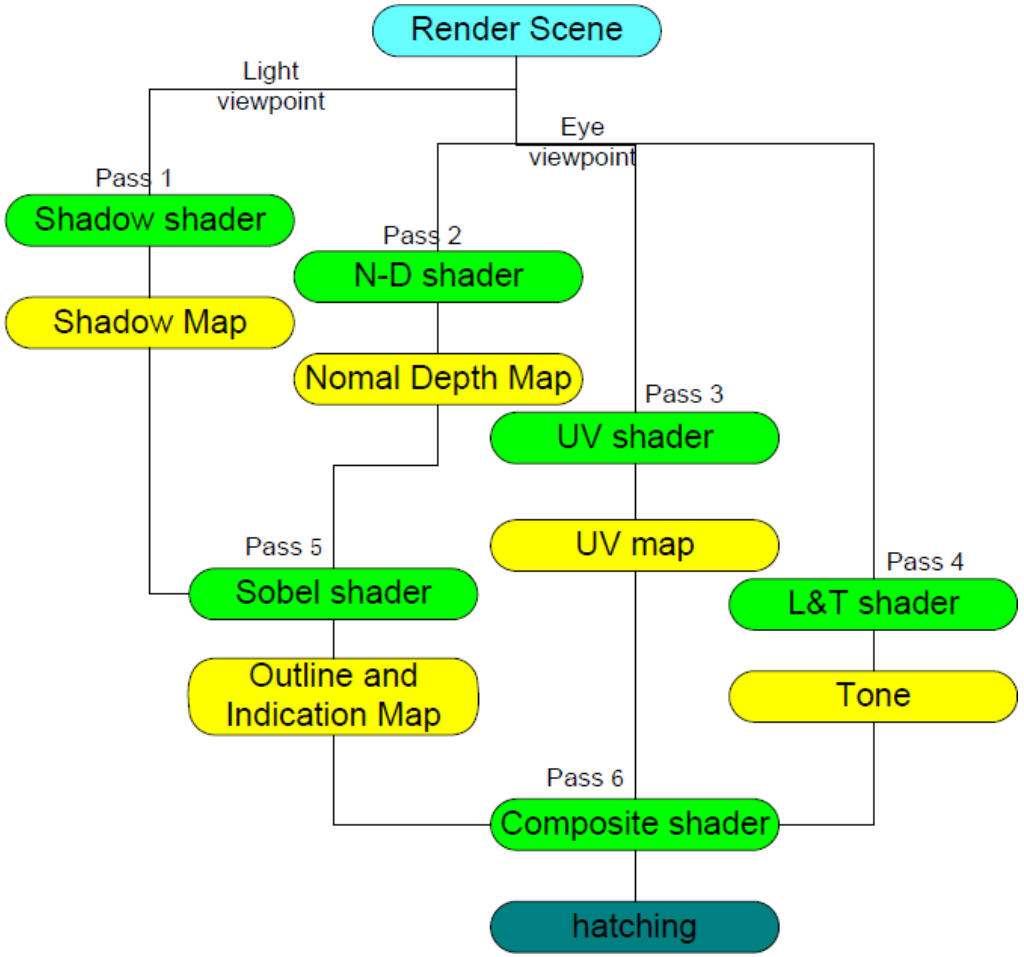




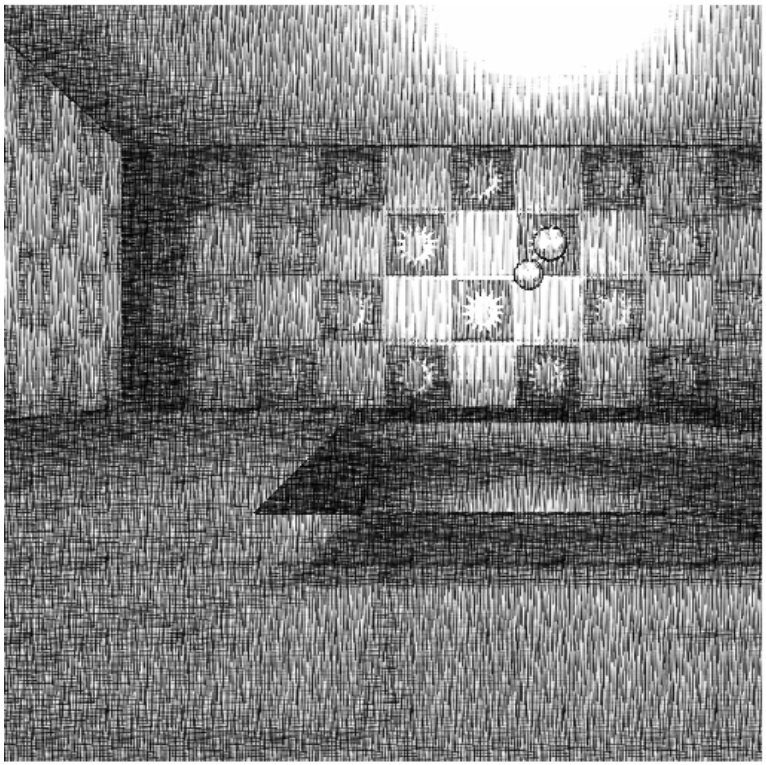
Direction of strokes

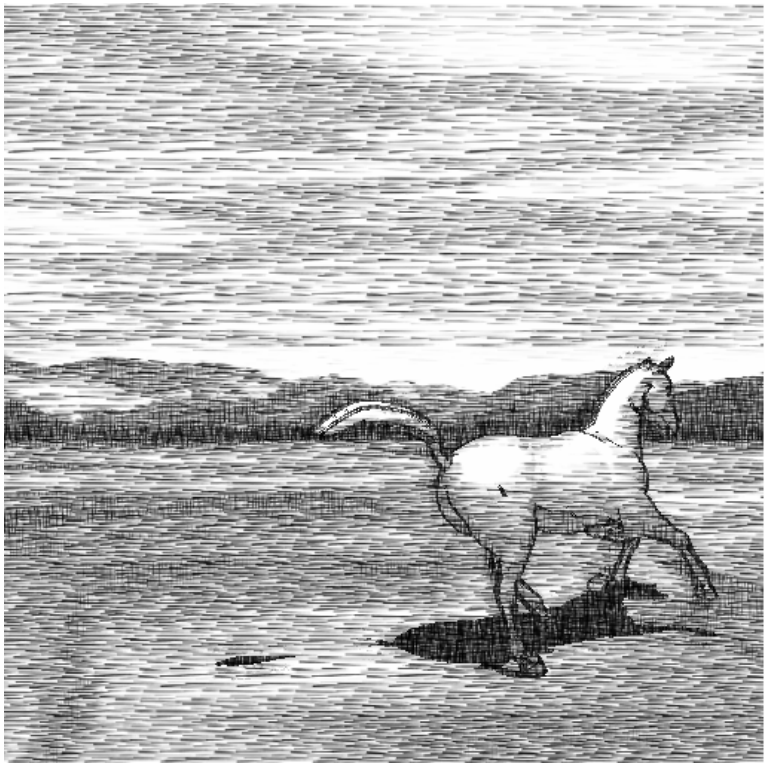


Hardware Hatching Process



Hardware Hatching Results



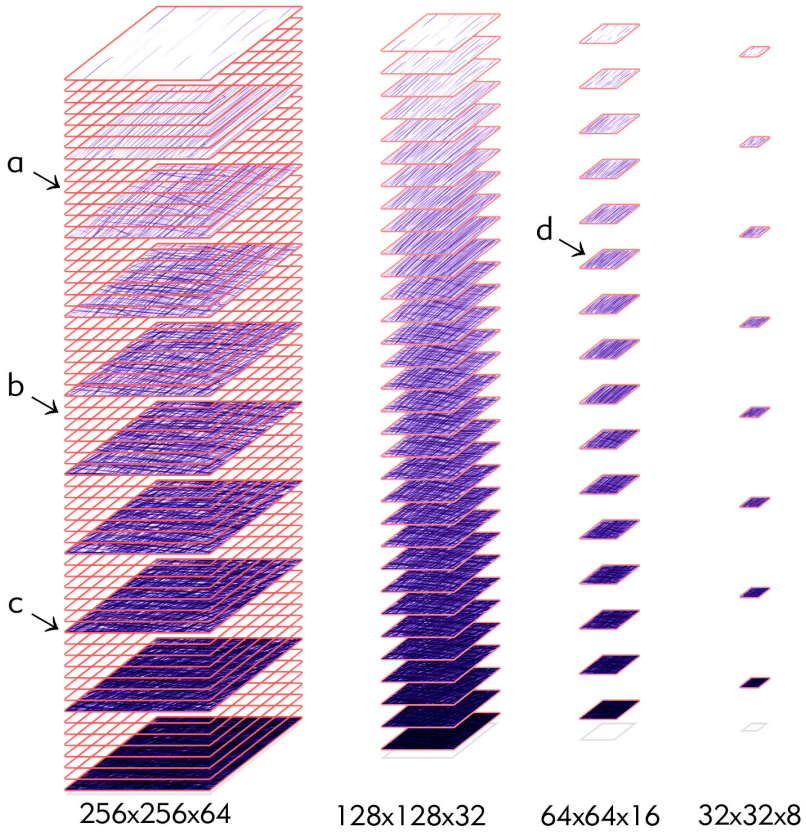


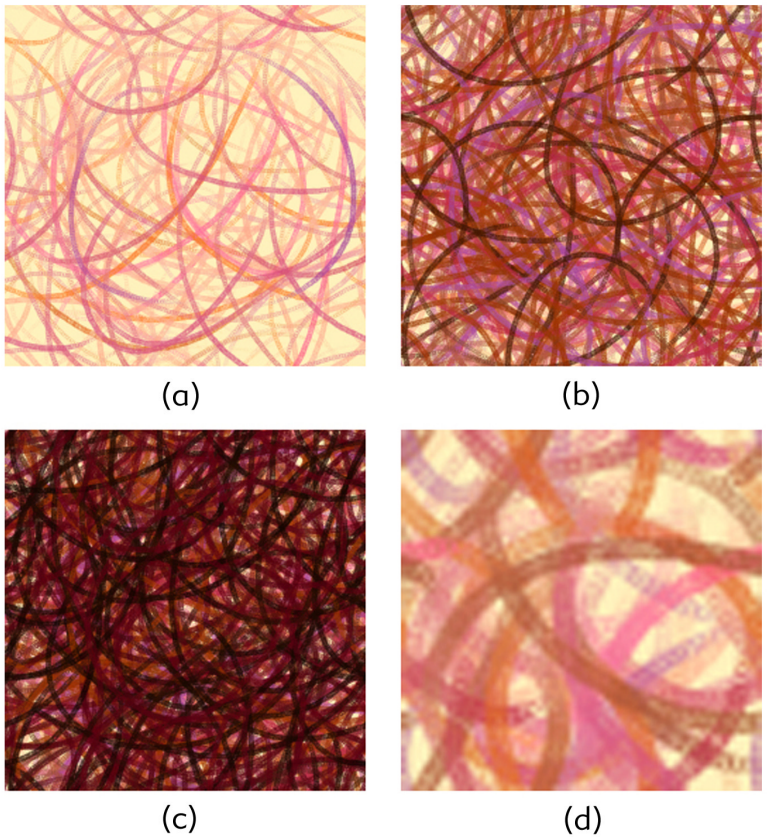
Video (http://www.youtube.com/watch?v=TUwY4D5cl6o)

Another hatching approach

Use tonal art maps, stored in volume textures.

The third coordinate is used to encode tone.





Another hatching approach

Another hatching approach

Video

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

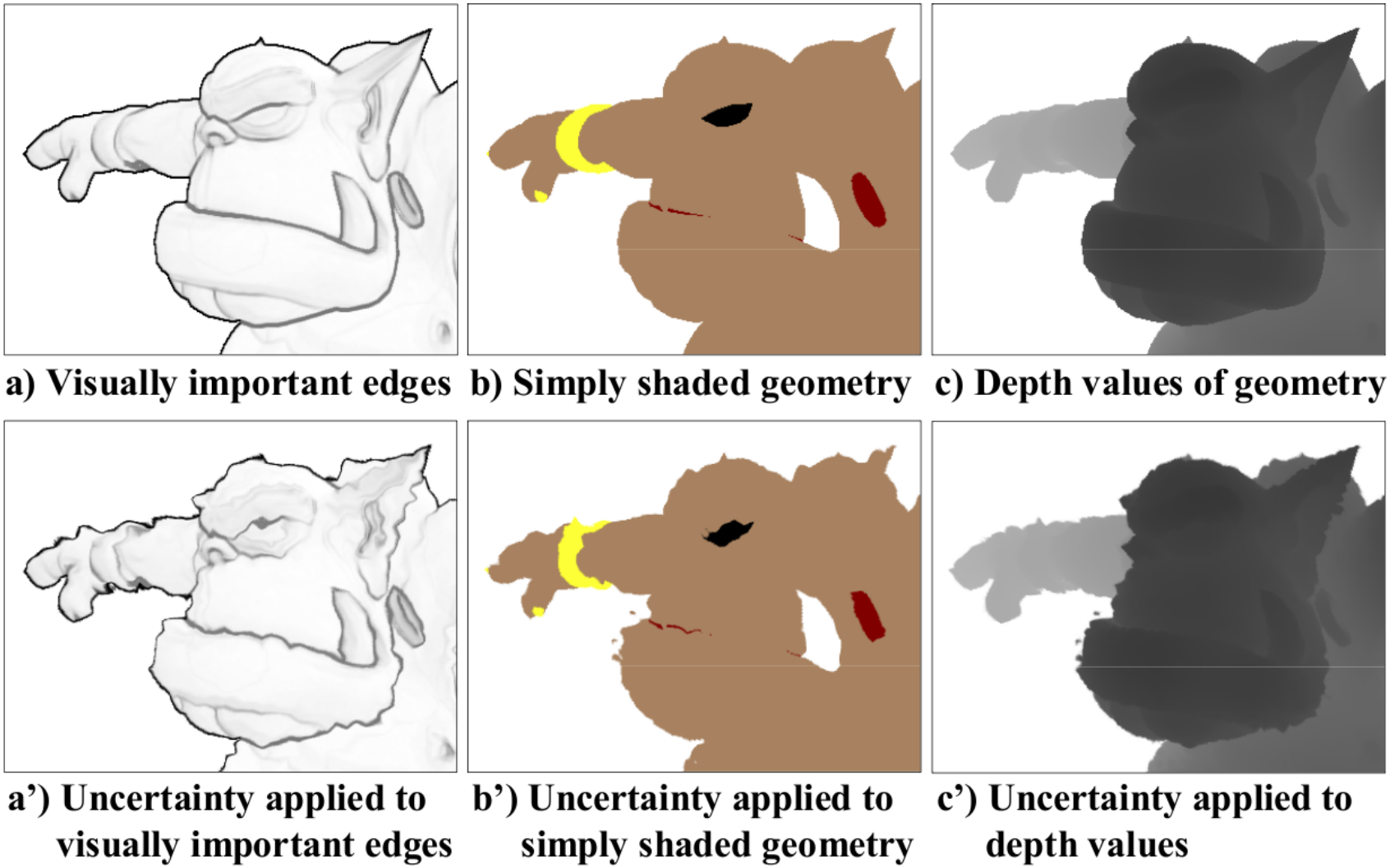
- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

Sketchy drawings

Try to create sketchy looking drawings
Method:

- Compute important edges
- Save them to a texture T_{Edge}
- Color surfaces
- Save result to texture $T_{Surface}$
- Draw depth buffer
- Save result to texture T_{Depth}
- Apply T_{Edge} and $T_{Surface}$ to a screen filling quad
- Apply noise in mapping to get perturbations
- Use T_{Depth} to retain depth information

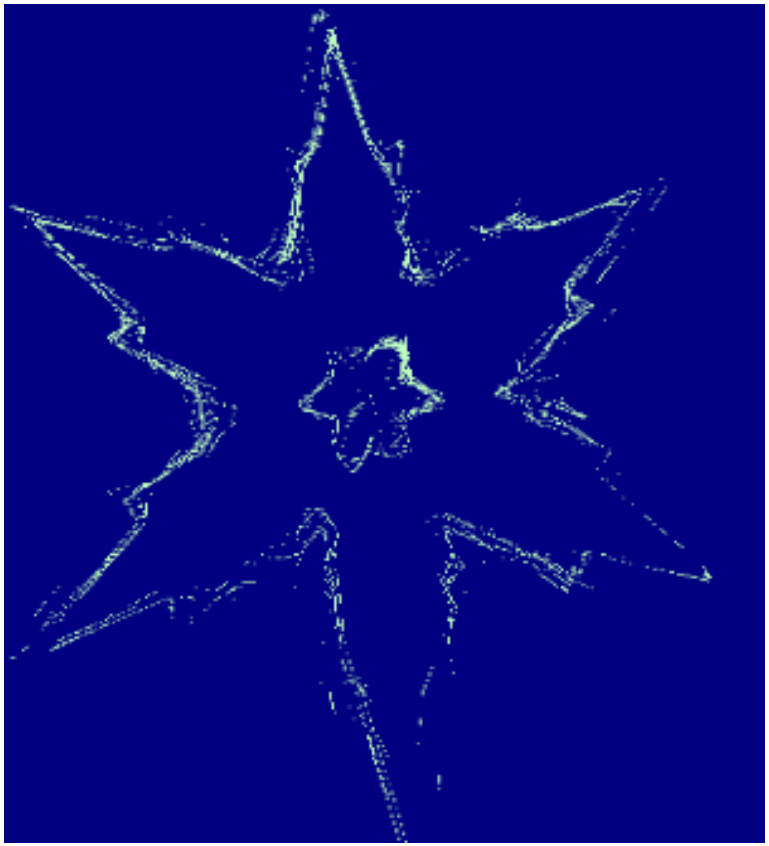
Sketchy drawings



Sketchy drawings

Sketchy drawings

(Created with a different method)

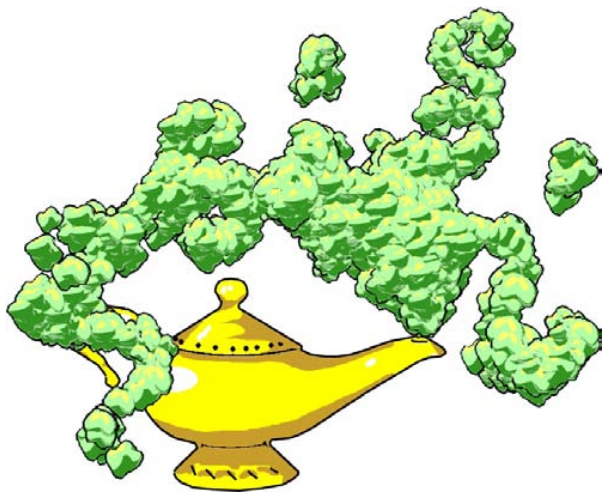


Example: Cartoon Style Smoke

Rendering animated cartoon style smoke and clouds with

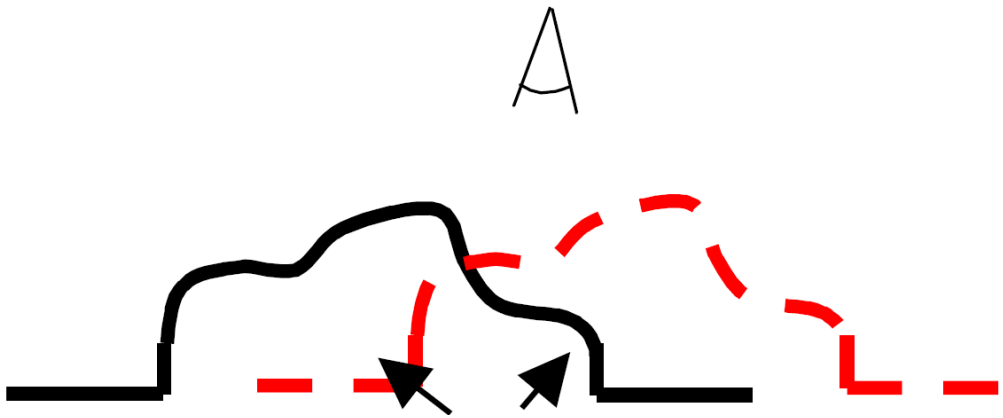
- Cel shading
- Outlines
- Self shading

Based on the paper by McGuire, Fein.



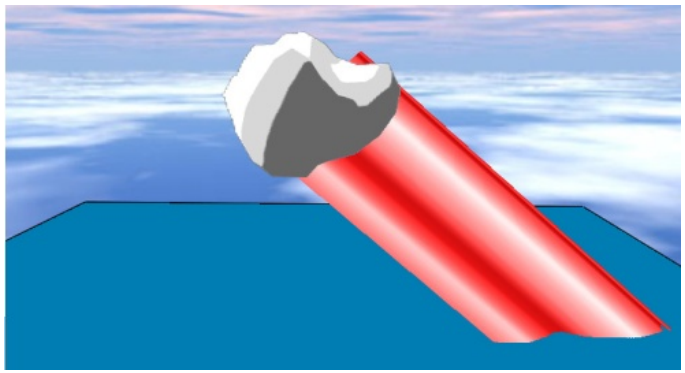
Cartoon Style Smoke - Process

- Simulate movement of smoke by a fluid simulator
- Draw smoke particles as nailboards
  - Apply shading to the particle
  - Create borders
- Use shadow volumes for self-shadows



Nailboards

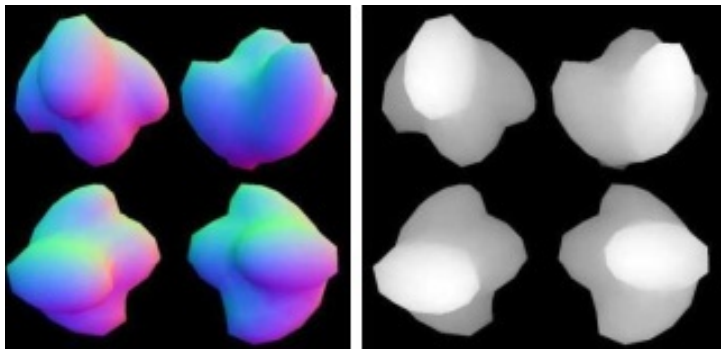
- Extension of a billboard
  - View aligned textured plane
  - Also called sprite
  - Extended with depth map
  - Introduced by Schauffler, 1997
- Computed from mesh prior to rendering
- Precomputed data stored in textures
- Defines a shadow volume
  - Optimization and extension of stencil shadow volumes



Precomputed data

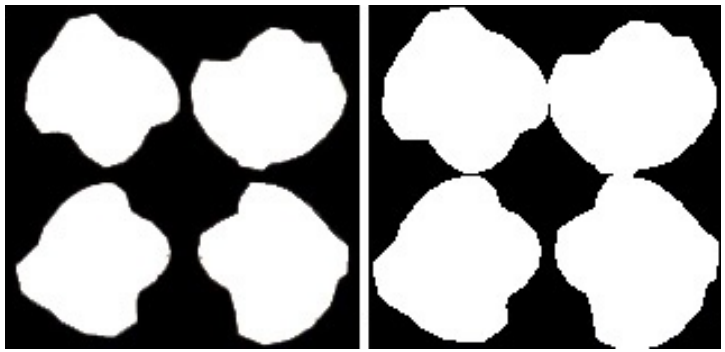
First texture:

- Surface normal
- Depth value

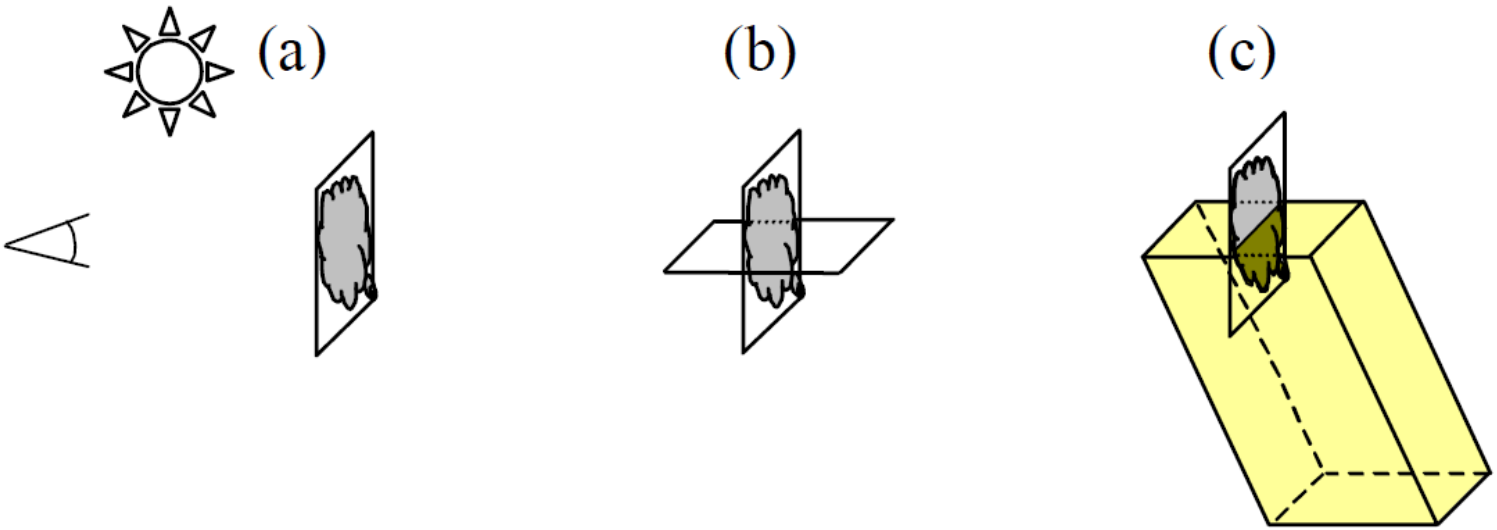


Second texture:

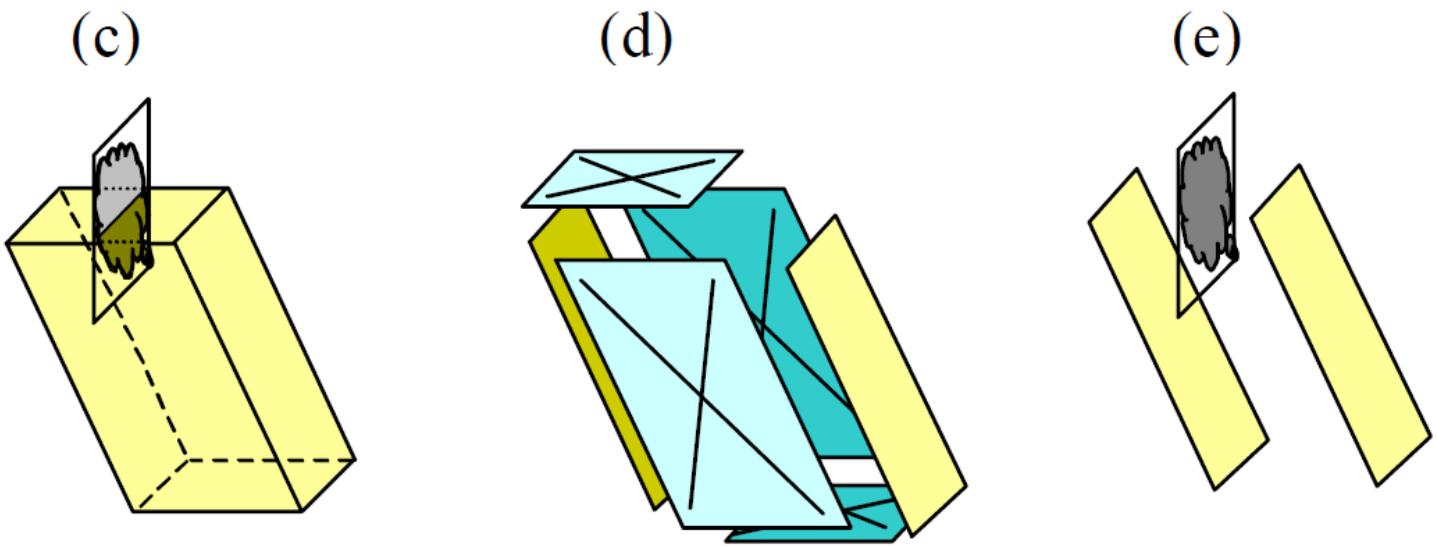
- Diffuse color
- Coverage mask



Shadow Volume - Computation (1)



Shadow Volume- Computation (1)



Cartoon Style Smoke - Shaders

The vertex shader:

- is used to created the view aligned nailboards
  - Expand points in the viewing plane
  - Movement vector determined by texture coordinate
- Determine camera-space normal

The fragment shader:

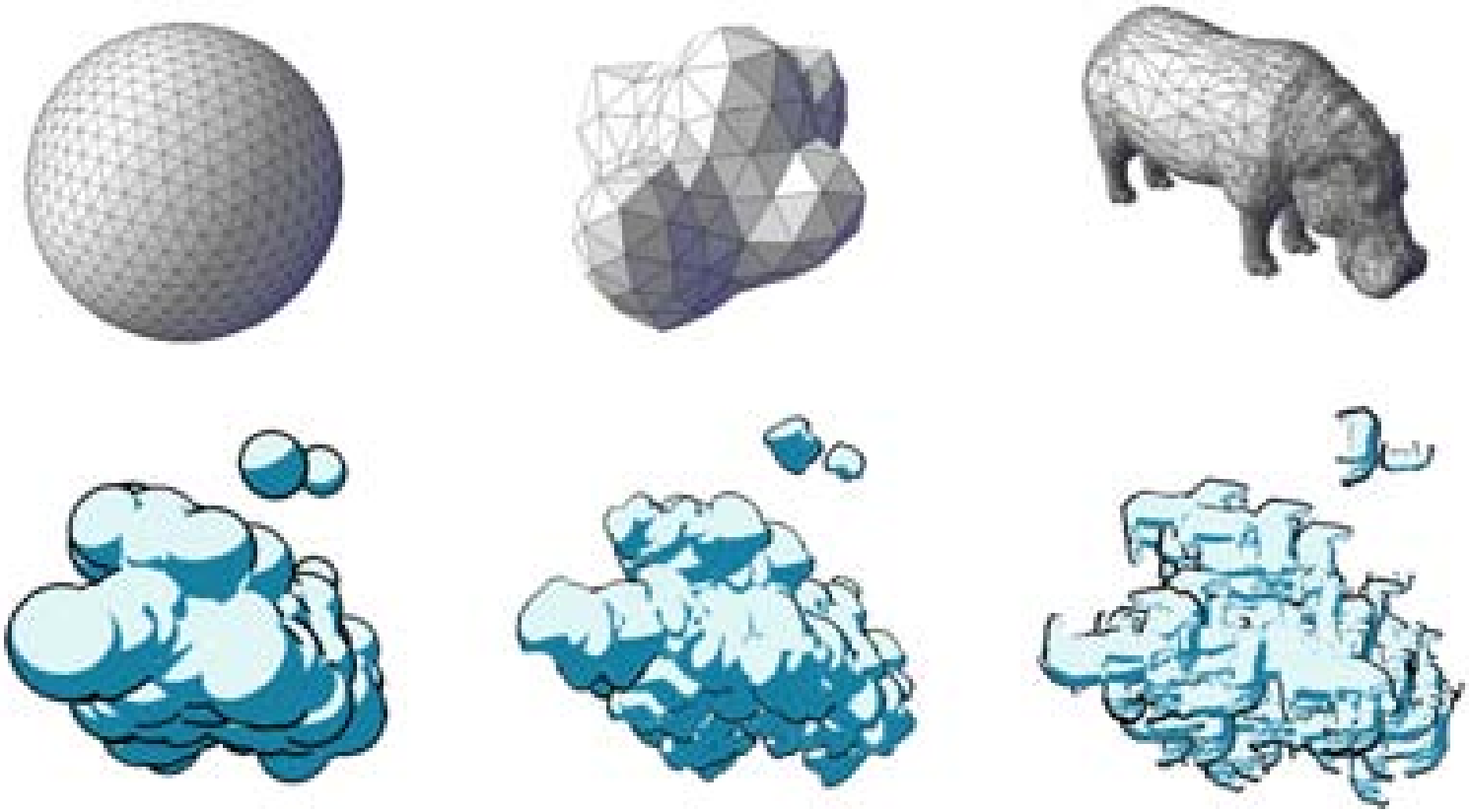
- Computes camera space depth
- Pixel color using Lambertian shading

Lambertian Shading

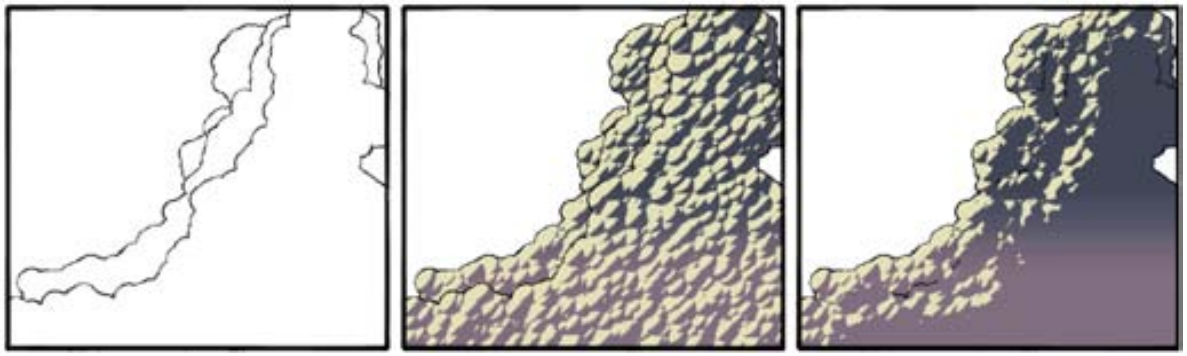
$$color = K_a(x, y) + C \cdot q(\max(0, N \cdot L))$$

- K_a is the ambient light color
- K_L is the light color
- N is the normal on the object
- L is the direction to the light source
- C is the diffuse term from the texture map
- q is a quantizing function
  Implemented as a 1-D texture

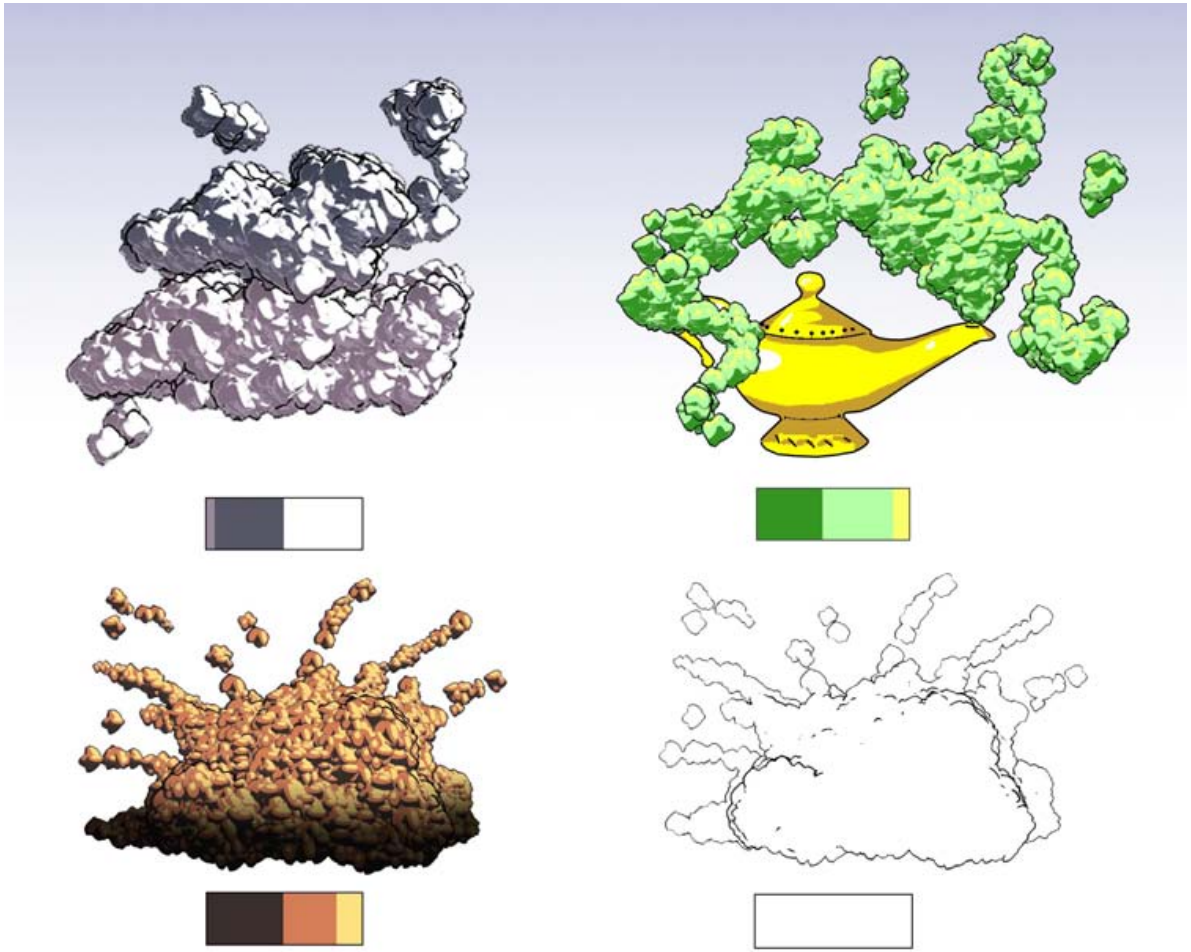
Cartoon Style Smoke - Results



Cartoon Style Smoke - Results



Cartoon Style Smoke - Results



Video

Graphics Hardware and NPR

What can we do?

- With shaders we can modify the rendering process
- We can extract additional data using multiple passes
- Using this data and appropriate shaders,
  we can create pretty much everything we want
- Shaders were introduced to create more realistic effects
- Since we can program them how we want we can get very different effects

Video: All kinds of shaders
(http://www.youtube.com/watch?v=JbooDujBEEY)

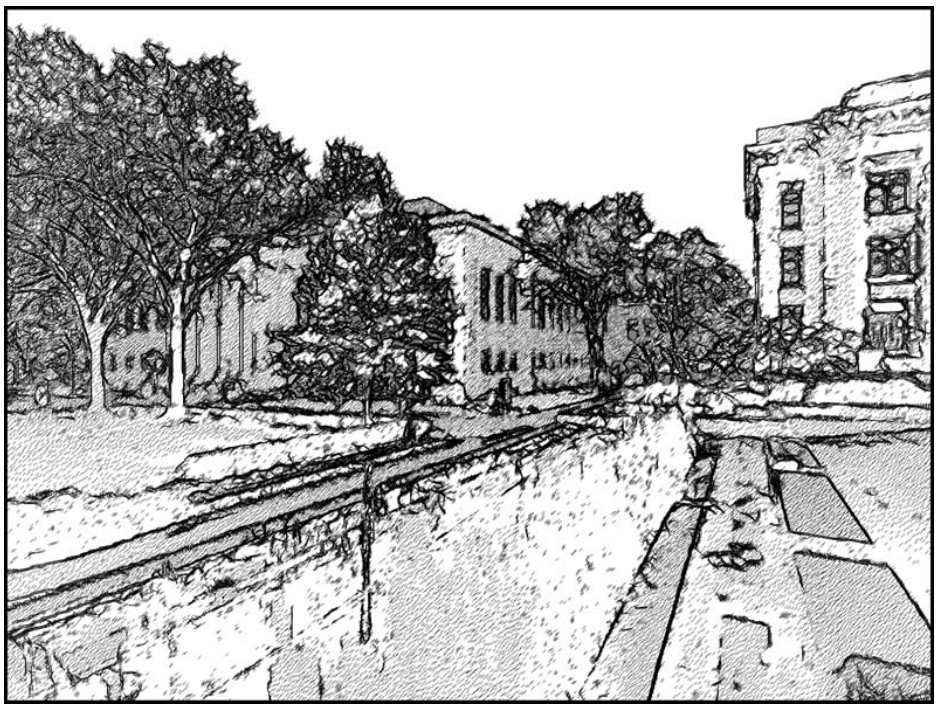
Graphics Hardware - Improvements

Improvements for NPR:

- Geometry shaders
  - Allow control over basic geometry
  - Works with graphics primitives
  - Has the ability to insert extra edges, which the vertex shader cannot do
  - Hence, the amount of data sent to the GPU can be reduced
- Faster or better algorithms

Concluding

- Graphics Hardware becomes more and more programmable
- We can use this to go beyond traditional rendering



Questions?