

NVIDIA™

GPU Toon Shading

D. Sim Dietrich Jr.

NVIDIA Corporation

sim.dietrich@nvidia.com

Summary

- **Non-Photorealistic rendering – ‘Toon Shading’**
  - **At no CPU cost**
  - **Requires DOT3 texture blending**
  - **Leverages 2 Types of TexGen**
  - **Leverages Projected Textures**
  - **Employs Cube Maps**



NVIDIA

Toon Shading

- Toon Shading is a type of NPR
- Black outline around object
- Flat or limited color shading
- Easy to get it working, hard to minimize the CPU impact



Black Outline

- The black outline around an object helps separate the object from the background and gives it that hand-drawn look
- We want a CPU-independent way to generate a black line around the edge of the object
- Drawing Lines doesn't count – Many cards emulate lines with Screen-Space Quads
  - Hardly CPU independent



TexGen Method

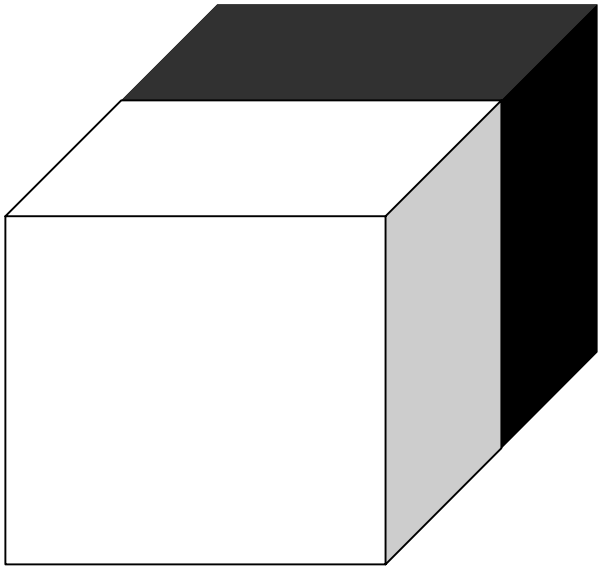
- One approach is to have a texture that has black and white parts
- The vertex normal is used to dot with the normalized vector from the vertex to the viewer
  - Did you say normalize?
  - Similar to Backface culling
- This dot product is used to index into a texture where black indicates a ~90 degree angle between the view vector and the vertex normal
- Obviously, this is not CPU independent, although it is still a valid technique



NVIDIA.

Cube Map Method

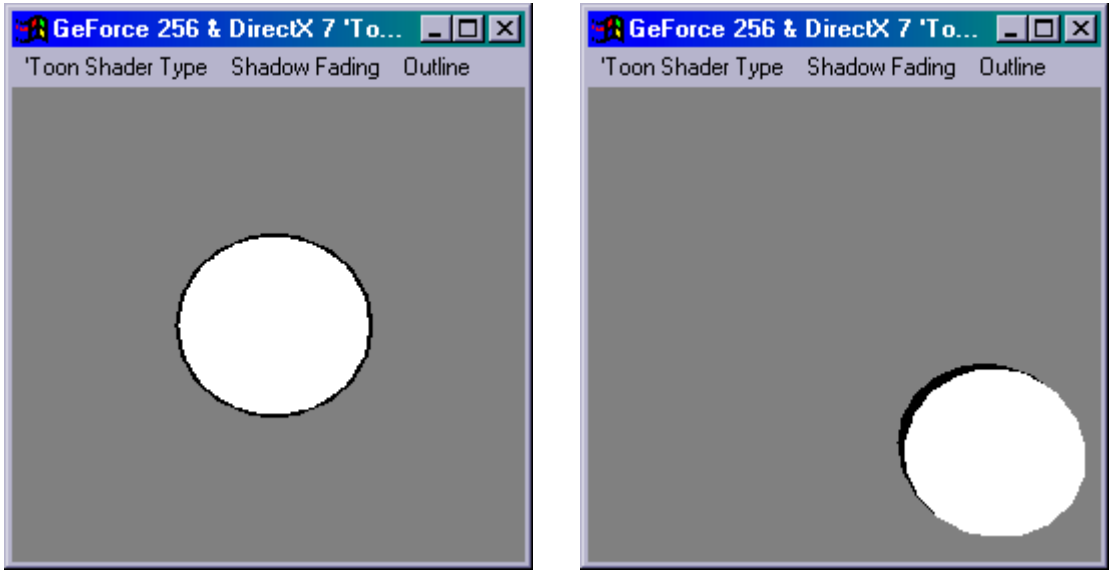
- Another approach is to store a black stripe in a cube map at the 90 degree angle
- By using the vertex normal in view space to index the Cube map, you get a black outline



NVIDIA.

Cube Map Method

- **Unfortunately, this method is only valid for orthographic projections**
- **Note how the outline warps as the ball moves to the corner of the viewport**



NVIDIA

Two-way TexGen?

- **The fundamental problem is that we need to take into account both the vertex normal** and **position.**

- **Texgen can handle only one**
  - **D3DTCI_CAMERASPACENORMAL**
- **or the other**
  - **D3DTCI_CAMERASPACEPOSITION**
- **…not both**



Dot product?

- The problem we are trying to solve is related to backface culling
- Back face culling requires taking both a vertex position and normal into account
- That means the operation boils down to a dot product
- Perhaps we can use D3DTOP_DOTPRODUCT3 texture blending…



NVIDIA

Using DOT3 for Toon Rendering

- We need to perform a dot product of the view vector with the perspective-adjusted normal, but we can't have a perspective-adjusted normal
- Since we can't adjust the normal based on vertex position, we can simply adjust the view-vector instead
- We need a way to adjust the view vector on a per-vertex basis



NVIDIA

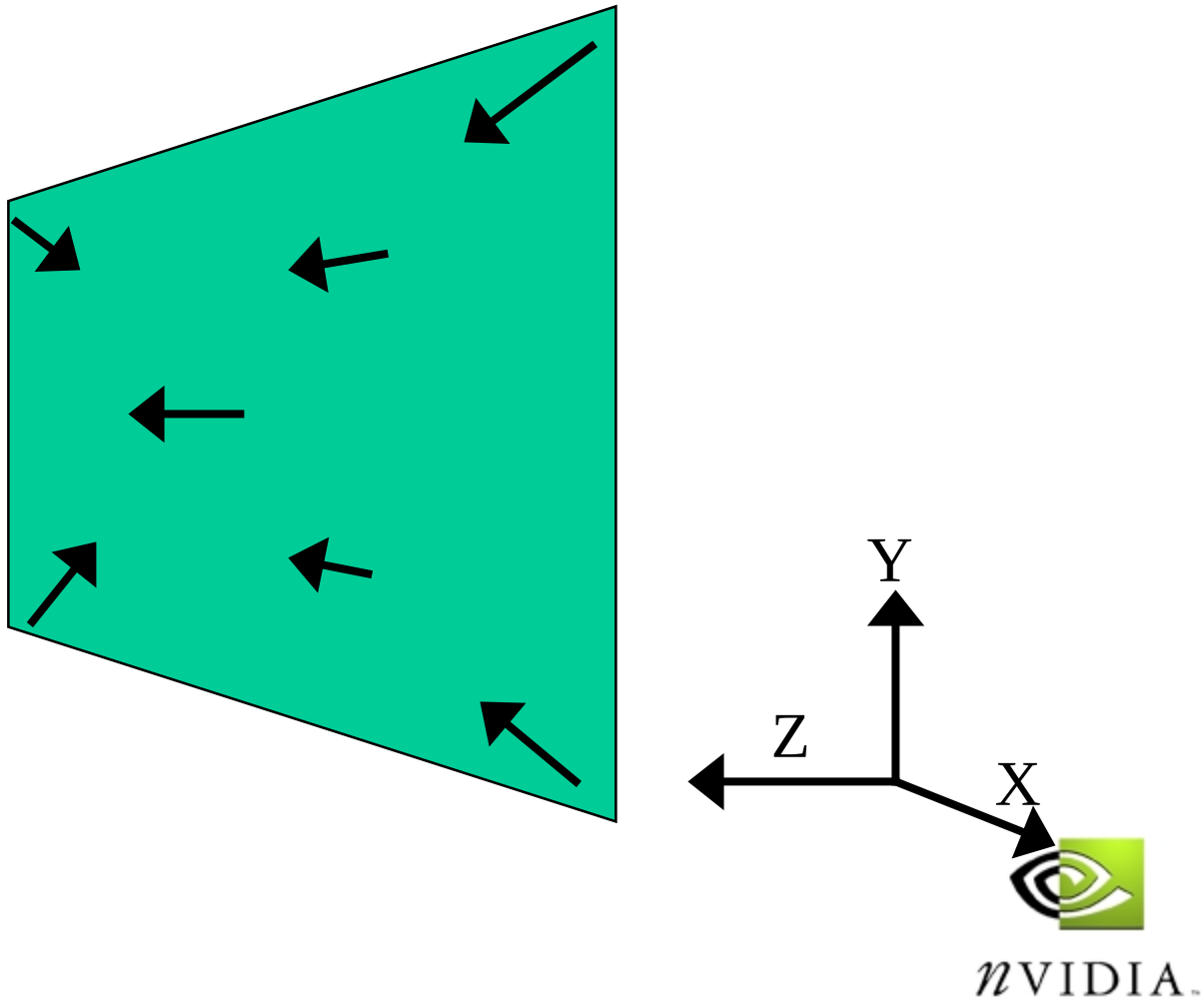
Adjusting the View Vector

- The idea is rather than using <0,0,1> as our constant view vector, we want a view vector that adjusts for perspective based on vertex <x,y,z> within the scene
- We can use D3DTCI_CAMERASPACEPOSITION texgen to get <x,y,z> view space coordinates
- And then use these to index into a screen-aligned projected texture



NVIDIA

Screen Aligned Projective Texture



NVIDIA.

Generating the Projective Texture

- This texture represents perspective-adjusted view vectors
- For each point on the screen, store the normalized vector from the point on the near clip plane to the origin
- The values at the edge of the texture are dependent on the field of view
- With a 90 degree field of view, the edge vectors will be be make ±45 degree angle with the original view vector
  - Thus changing from –45 to +45 across the texture



Using the Projective Texture

- This technique for Toon rendering requires taking a dot product between two normalized vectors
- The first vector is a the vertex normal
  - Its RGB value comes from the vector normalization cube map
- The second vector is the projected vertex position, < X / W, Y / W >
  - Its RGB value comes from the screen-aligned projective texture



NVIDIA

Texture Stage Setup

- So, we perform
  - TEXTURE0 // vertex normal from // cubemap
  - SELECTARG1
  - CURRENT // vertex normal
  - DOT3 // dot product
  - TEXTURE1 // perspective-adjusted // view normal



Render State Setup

- Now that we have a dot product, what do we use it for?
  - We don’t use the Grayscale RGB
  - We use the Alpha instead
- D3DTOP_DOTPRODUCT3 replicates the result, clamped from [0..1] into RGB and Alpha
- We can use Alpha test to decide if a particular pixel falls on the outline or not



NVIDIA.

Alpha Testing The Outline

- We can set the Alpha Reference value to be a small number, like 0x1d
- We set the Alpha Test to throw away anything greater than the reference value, by setting the Alpha Compare Function to D3DCMP_LESSEQUAL
- This draws only the outline
- Set Alpha Blending to True
- Set the Blend mode to D3DBLEND_ZERO, D3DBLEND_ZERO



NVIDIA.

Putting It Together

Firstly, the Color Pass :

- **First, render the object, using either its diffuse colors, lighting, or a flat color with D3DTA_TFACTOR**
- **This puts the base color(s) of the object into the frame buffer**
  - **With no outline yet**
  - **We use another pass to draw the outline**



Putting It Together

Now for the Outline Pass :

- **Set the Z Compare to D3DCMP_EQUAL**
- **Select the Normalization Cube Map into Texture 0, the Projected Texture into Texture 1**
- **Set up the alpha test and alpha blend for the black outline**



NVIDIA

Putting It Together (cont.)

- Set up the two Texgen modes
  - Cubemap's Texgen should be D3DTCI_CAMERASPACENORMAL
  - View Vector Texture should be D3DTCI_CAMERASPACEPOSITION
- Turn on projection for Texture 1
- Set up the Dot Product in the texture combiners



NVIDIA.

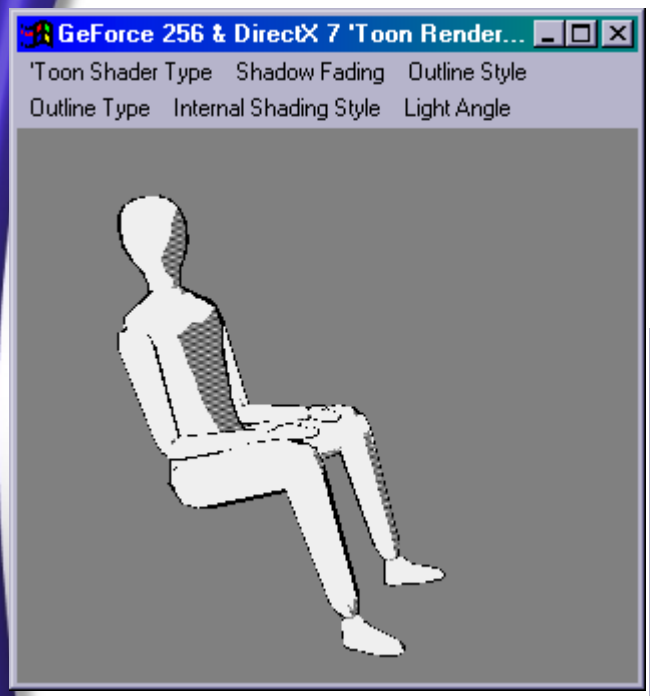
Time To Draw

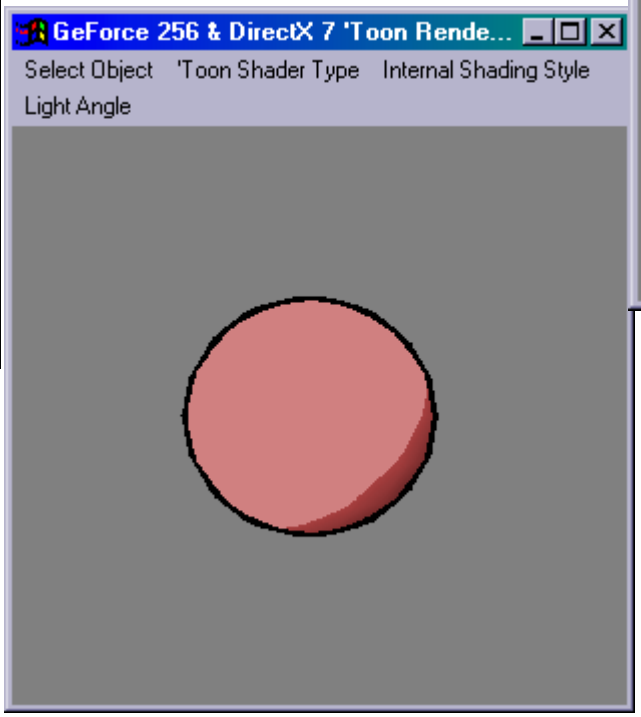
- **Now Draw the object again**
  - **This time the black outline will be overlaid on the object's silhouette edges**
- **You can can ensure a consistent outline thickness by adjusting the Alpha Reference value up and down based on the distance of the object**



NVIDIA.

Examples







NVIDIA.