Mathematics of Per-Pixel Lighting

Cass Everitt
NVIDIA Corporation
cass@nvidia.com

Overview

- Why Per-Pixel Lighting?
- Review
  - OpenGL Transforms and Spaces
  - OpenGL Per-vertex Lighting
  - *Object Space* Per-vertex Lighting
- ***Surface-local Space*?**
  - Other names
  - Why is this necessary?
  - *Surface-local Space* Per-Vertex Lighting
- Per-Pixel Lighting
  - In *Surface-local Space*
  - In other spaces?

Why Per-Pixel Lighting?

- Because it looks better than per-vertex lighting
- Because it's hardware accelerated
- Because everyone else is doing it
  - Don't be the last on your block

This is do-it-yourself lighting

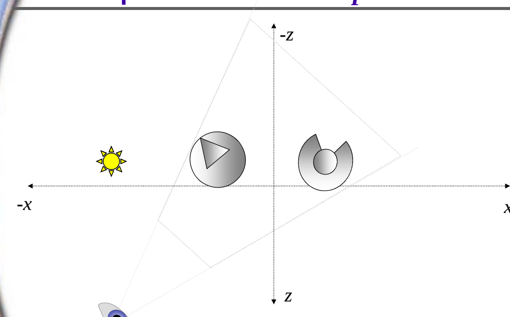
- You get total control, but this means you have to do it all
  - No `glShadeModel(GL_PHONG)`
  - No `glEnable(GL_BUMP_MAPPING)`
- If you don't know how to implement per-vertex lighting, learn how to do that first
- Per-pixel shading is an extension of per-vertex shading (for the most part)

OpenGL Transformations

object space
↓ MODELVIEW matrix
eye space
↓ PROJECTION matrix
clip space
↓ Perspective Divide
normalized device coordinates
↓ viewport/depthrange scale & bias
window space

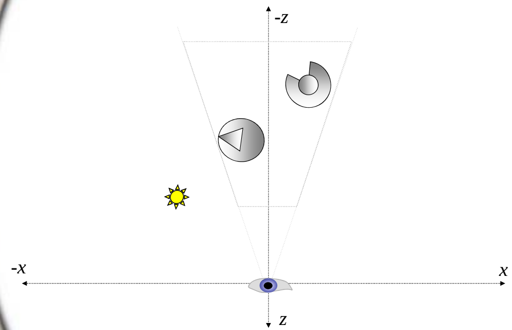
- OpenGL operation transforms coordinates through several *coordinate frames* or *spaces*
- Each of the *spaces* has various properties that make it useful for some operation
- Vertex attributes are specified in *object space*
- Lighting, eye-linear texgen, and fog happen in *eye space*
- Clipping happens after projection in *clip space*
- Rasterization happens in *window space*

Example Scene -- *world space*



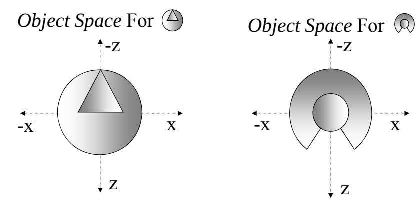
Note: *world space* is not an explicit space in OpenGL

Example Scene -- *eye space*



7

Object Space



Each object has its own origin, orientation, and scale

8

OpenGL Per-Vertex Lighting

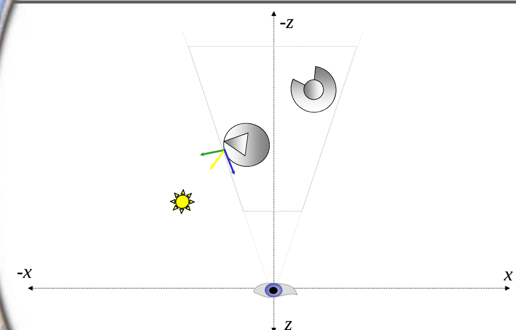
- **For OpenGL Per-Vertex Lighting, all calculations happen in** *eye space*
- **Not essential, but convenient**
- **For each OpenGL per-vertex light, the illumination is computed as (assuming separate specular)**

$$C_{pri} = (spot)(att)\left[a_{cm}a_{cli} + (\mathbf{n} \bullet \mathbf{l})d_{cm}d_{cli}\right]$$

$$C_{sec} = (spot)(att)(f)(\mathbf{n} \bullet \mathbf{h})^{s_{rm}} s_{cm}s_{cli}$$

9

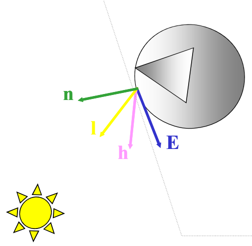
Lighting in *eye space*



10

Lighting in *eye space* (2)

The vectors…



11

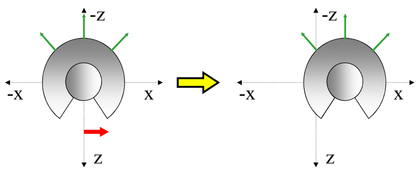
Transforming Normals

- **To evaluate the lighting equation in** *eye space*, **normals must be transformed from** *object space* **into** *eye space*
- **Normals are not simply transformed by the modelview matrix like position**
- **You may know from the Red Book or various other sources that "normals are transformed by the inverse-transpose of the modelview matrix", but let's consider why…**
- **The following slides should help provide some intuition about the transforming of normals**

12

Transforming Normals (2)

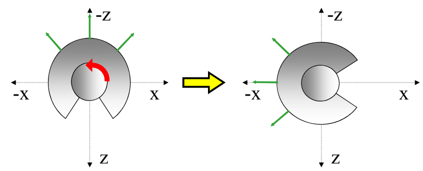
- **Translation of position does not affect normals**



13

Transforming Normals (3)

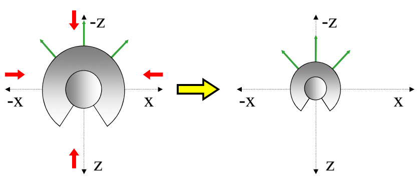
- **Rotation is applied to normals just like it is to position**



14

Transforming Normals (4)

- **Uniform scaling of position does not affect the direction of normals**

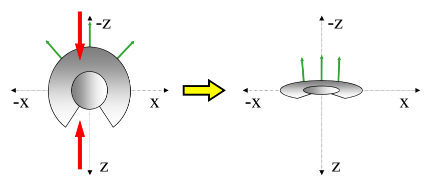


Note that we are *only* considering how the *direction* of a normal is affected by transforming the position, *not magnitude*

15

Transforming Normals (5)

- **Non-uniform scaling of position does affect the direction of normals!**
  - ***Opposite*** **of the way position is affected – or the** ***inverse*** **of the scaling matrix that's applied to position**



Note that we are *only* considering how the *direction* of a normal is affected by transforming the position

16

Transforming Normals (6)

- **To summarize, these are the basic position transformations and the corresponding normal transformation:**

	position	normal
translation	$\mathbf{T}$	$\mathbf{I}$
rotation	$\mathbf{R}$	$\mathbf{R}$
scaling	$\mathbf{S}$	$\mathbf{S}^{-1}$

- **Note that any sort of scaling applies inversely to the normal – we treat all scales (uniform and non-uniform) the same**
- **This is why we need GL_NORMALIZE and GL_RESCALE_NORMAL for OpenGL lighting**
- **We have to deal with it in per-pixel lighting as well**

17

Transforming Normals (7)

- **How does this match what OpenGL does?**

$$\boldsymbol{n}_e = \mathbf{M}^{-\mathrm{T}} \boldsymbol{n}_o$$

- **For simplicity, consider $\mathbf{M}$, the modelview matrix, is composed of a scale and a rotation**
  - *inverse-transpose* **is distributive**
  - **For rotation (orthonormal) matrices** $\mathbf{R}^{-1} = \mathbf{R}^{\mathrm{T}}$**, and** $\mathbf{R}^{-\mathrm{T}} = \mathbf{R}$
  - **For scaling (diagonal) matrices** $\mathbf{S} = \mathbf{S}^{\mathrm{T}}$

$$\begin{aligned} \mathbf{M}^{-\mathrm{T}} &= (\mathbf{RS})^{-\mathrm{T}} \\ &= \mathbf{R}^{-\mathrm{T}}\mathbf{S}^{-\mathrm{T}} \\ &= \mathbf{RS}^{-1} \end{aligned}$$

This matches our ad hoc result!

18

Object Space Per-Vertex Lighting

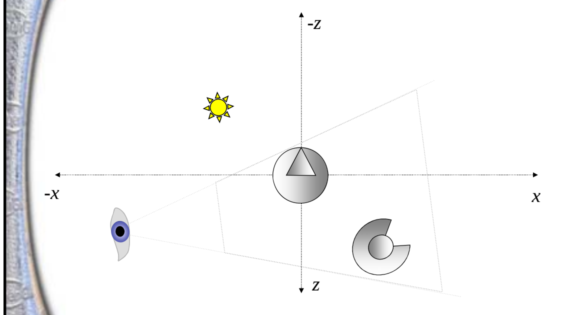
- Nothing in the lighting equation requires evaluation in *eye space* - consider lighting in *object space* instead
  - Non-uniform scaling in the modeling matrix would complicate things, so we will ignore that for now…
- If the modeling matrix is simply a rigid body transform, then this is easy…
  - Need to transform the light into *object space* from eye space

    $l_{obj} = \mathbf{M}^{-1} l_{eye}$ local light source

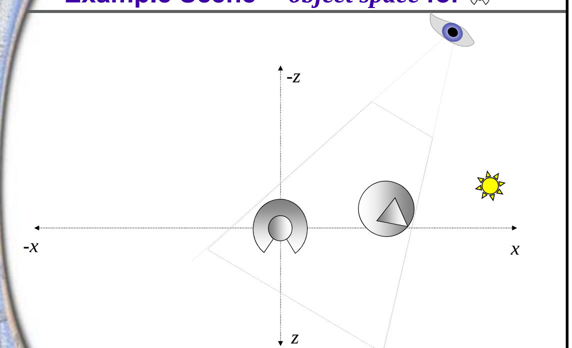
    $l_{obj} = \mathbf{M}^{T} l_{eye}$ infinite light source

  - No need to transform each normal now (cheaper)

Example Scene -- *object space* for

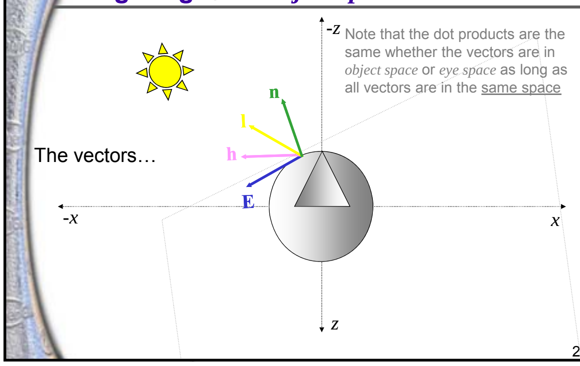


Example Scene -- *object space* for



Lighting in *object space*

Note that the dot products are the same whether the vectors are in *object space* or *eye space* as long as all vectors are in the same space



Surface-local Space

- *surface-local space*
- **surface-local matrix**
- *object space*
- **MODELVIEW matrix**
- *eye space*
- **PROJECTION matrix**
- *clip space*
- **Perspective Divide**
- *normalized device coordinates*
- **viewport/depthrange scale & bias**
- *window space*

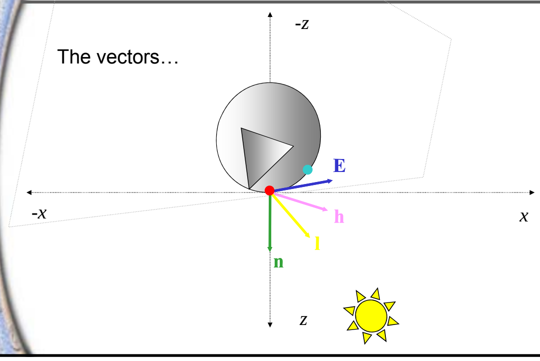
- This gets called a lot of things…
  - *surface-local space*
    - *tangent space*
    - *texture space*
- A *surface-local space* is a class of spaces defined for every point on a surface
- *Tangent space* and *texture space* are *surface-local spaces* that give specific definitions to the basis vectors
- Consider one additional transform from *surface-local space* to *object space*

Surface-local Space (2)

- The classes of *surface-local space* we use are defined for every point on a surface such that the point is at the origin, and the geometric surface normal is along the positive z axis
  - Note that for per-pixel lighting the geometric surface normal is generally ***not*** what we use in the lighting equation
- The x and y axes are orthogonal and in the tangent plane of the surface
- Now the entire scene can be defined relative to any point on any surface in the scene – not just relative to any object

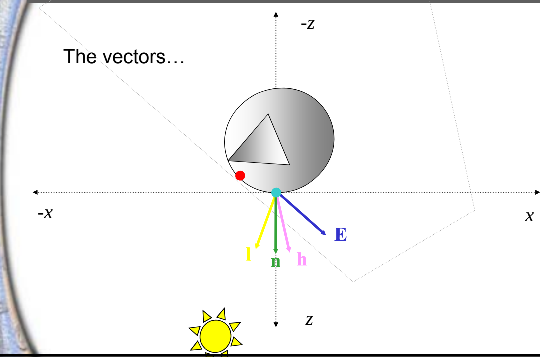
Lighting in *surface-local space*

The vectors…



Lighting in *surface-local space*

The vectors…



Surface-local matrix

- **If we specified vertices in** *surface-local space*, **they'd all be the same!**
  - **glNormal3f(0,0,1); glVertex3f(0,0,0);**
- **The surface-local matrix, S_l, would provide the** *object space* **position and the** *object space* **normal orientation, and it would vary per-vertex:**

$$S_l = \begin{bmatrix} T_x & B_x & N_x & P_x \\ T_y & B_y & N_y & P_y \\ T_z & B_z & N_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T -- tangent vector
B -- binormal vector
N -- object space vertex normal
P -- object space vertex position

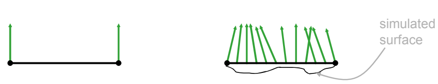
- **More on the tangent and binormal (*T* and *B*) vectors later…**

Per-Vertex Lighting in *surface-local space*

- **As with lighting in** *eye space* **or** *object space*, *surface-local space* **is a perfectly valid coordinate frame to evaluate the lighting equation**
- **We simply transform the light and eye into** *surface-local space* **– the normal is known by definition, so it doesn't need to be transformed**
- **Compare** *eye space* **and** *surface-local space* **lighting:**
  - *Eye space* **lighting: the light vector or eye vector are "free", but you must transform each normal into** *eye space*
  - *Surface-local space* **lighting: the normal is free, but you must transform the light and eye vectors into** *surface-local space*

Per-Pixel Lighting

- **Getting back to the original point…**
- **We really want to evaluate the lighting equation per-pixel**
- **Rather than passing in normals per-vertex, we'll fetch them from a texture map**
  - **We simulate surface features with illumination only**



per-vertex normals   per-pixel normals

Per-Pixel Lighting (2)

- **The texture map containing normals (normal map) clearly uses normals that are not aligned with the +z axis in** *surface-local space*
  - **This makes the tangent and binormal vectors important (see discussion later)**
- **GPUs certainly have enough horsepower to evaluate the illumination equation at each pixel – but it is more expensive in eye space!**
  - **That would require transforming each normal into eye space (<u>after</u> fetching it from the texture map)**

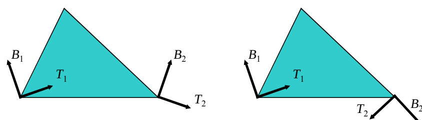
Per-Pixel Lighting (2)

- **The better solution is to light in** *surface-local space*
  - **Fetched normals are already in the correct space**
  - **Light and eye vector interpolate nicely as long as the tangent and binormal are "well behaved"**
- **All remaining arithmetic can be evaluated with register combiners** (not important these days)
- **Minor limitation: as with** *object space* **per-vertex lighting, you can't have a non-uniform scale without requiring a per-normal transform and renormalize**
  - **don't do lots of non-uniform scaling -- it won't behave correctly**

31

Tangent and Binormal

- **Whether we implement per-pixel lighting in** *surface-local space* **or** *eye space*, **the tangent and binormal vectors need to be well-behaved from vertex to vertex**
- **Specifically,** $\|lerp(a,T_1,T_2)\| \approx 1$
  and $\|lerp(a,B_1,B_2)\| \approx 1$

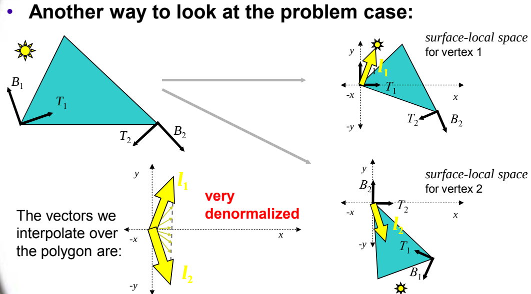


good bad

32

Tangent and Binormal (2)

- **Another way to look at the problem case:**



33

Tangent and Binormal (3)

- **In the previous case, we considered transforming the light into the** *surface-local space* **of each vertex and interpolating it for the per-pixel light vector -- this is what we would do for GeForce2** (old GPUs)
- **With modern GPus, we can interpolate the 3x3 matrix over the surface and transform the normals by it – for this case if the tangent and binormal are not well-behaved, other anomalous behavior will result**
  - **Normal "twisting"**
  - **Incorrect bump scale/smoothing**
  - **The interpolated matrix should be "nearly orthonormal"**

34

Questions?

Cass Everitt

cass@nvidia.com

www.nvidia.com/Developer

35