CS 418: Interactive Computer Graphics

Transforming Normals

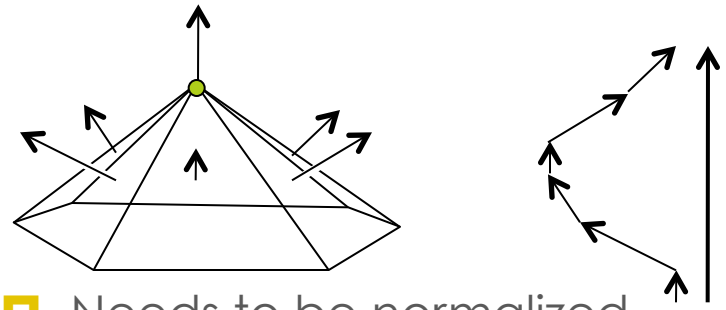
Eric Shaffer

Surface Normals

- Can be defined per face or per vertex
- Per face normal of a ccw face

$$\mathbf{n} = (\mathbf{x}_1 - \mathbf{x}_0) \times (\mathbf{x}_2 - \mathbf{x}_0)$$

  - Needs to be unitized before lighting
- Per vertex normal
  - Sum of normals of adjacent faces



  - Needs to be normalized

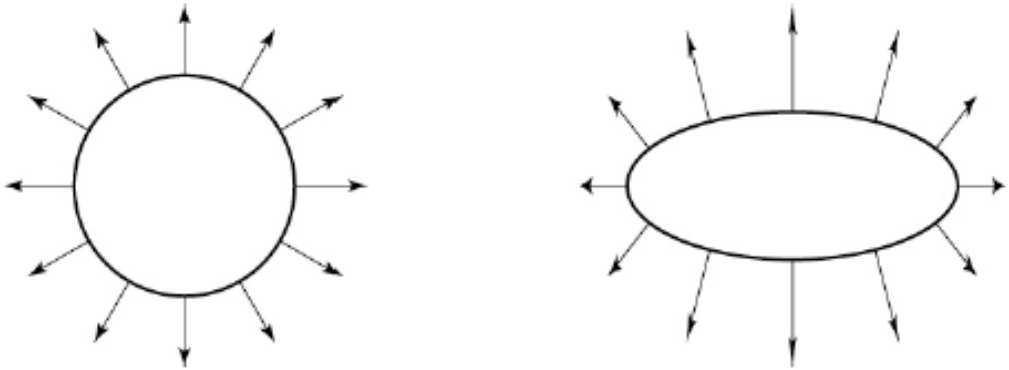
Transforming Normal Vectors

Affine transformations of differences of points work as expected

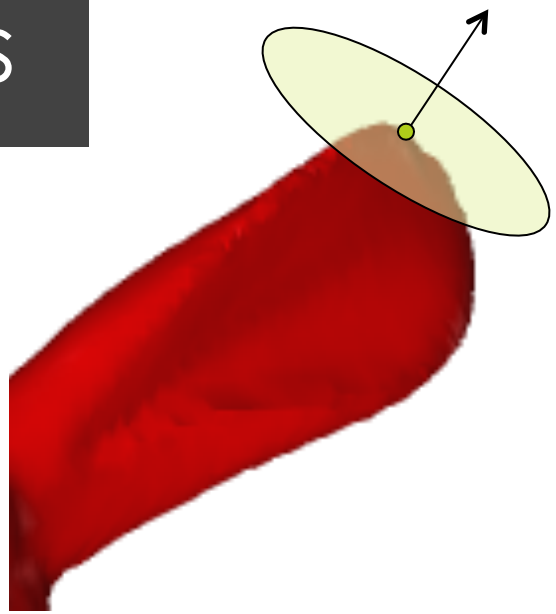
So affine transformations of tangents work

This is not true of normal vectors

We need to transform normals to correctly shade surfaces



Transforming Normals



- First order neighborhood of a point on a surface described by a tangent plane

- A tangent vector **t** at a point and the normal **n** are orthogonal
- So $\mathbf{t \cdot n = t^T n = 0}$
- This should be true of the transformed geometry as well
  - Let M be the modelview matrix
- So we seek a matrix X such that
  $\mathbf{(M\,t) \cdot (Xn) = (Mt)^T (Xn) = 0}$

  $\mathbf{(Mt)^T (Xn) = t^T M^T Xn = 0}$ if $\mathbf{M^T X = I}$

  So $\mathbf{X = (M^T)^{-1}}$

Computing the Inverse Transpose

- If your ModelView only uses uniform scaling and rotations and translations
  - you can transform normals by the top left 3x3 portion of the ModelView matrix
  - Why?

- Otherwise explicitly compute the inverse transpose
  - Only operate on the 3x3 portion (much faster that inverting 4x4)
    - Use a numerical library function to invert the matrix
  - Or keep track of the inverse transpose as you build the ModelView

- In either case:
  always normalize the normal to unit length afterwards