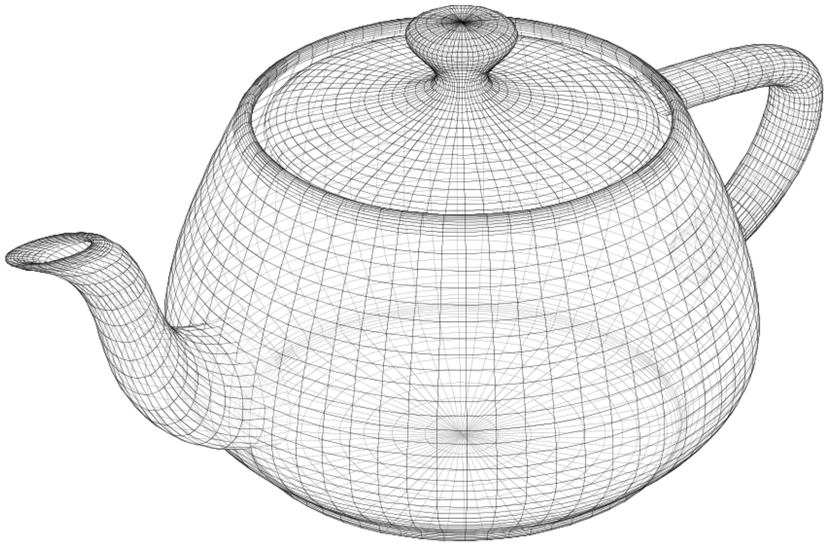


Computing and Transforming Surface Normals

CS 418: Interactive Computer Graphics

Professor Eric Shaffer

ILLINOIS

Normal Vectors on Surface Meshes

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

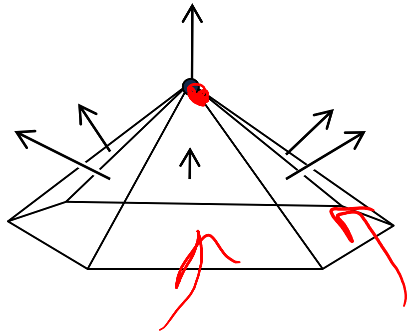
$$\mathbf{n} = (\mathbf{v}_1 - \mathbf{v}_0)\times(\mathbf{v}_2 - \mathbf{v}_0)$$

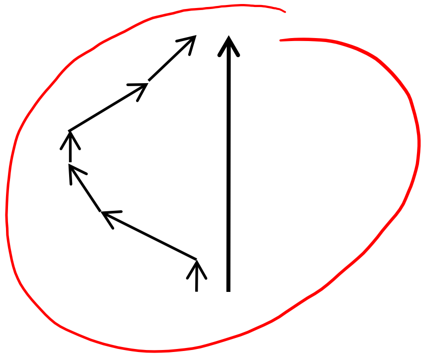
  - Needs to be unitized before lighting

ccw

per face

- Per vertex normal
  - Sum of normals of adjacent faces



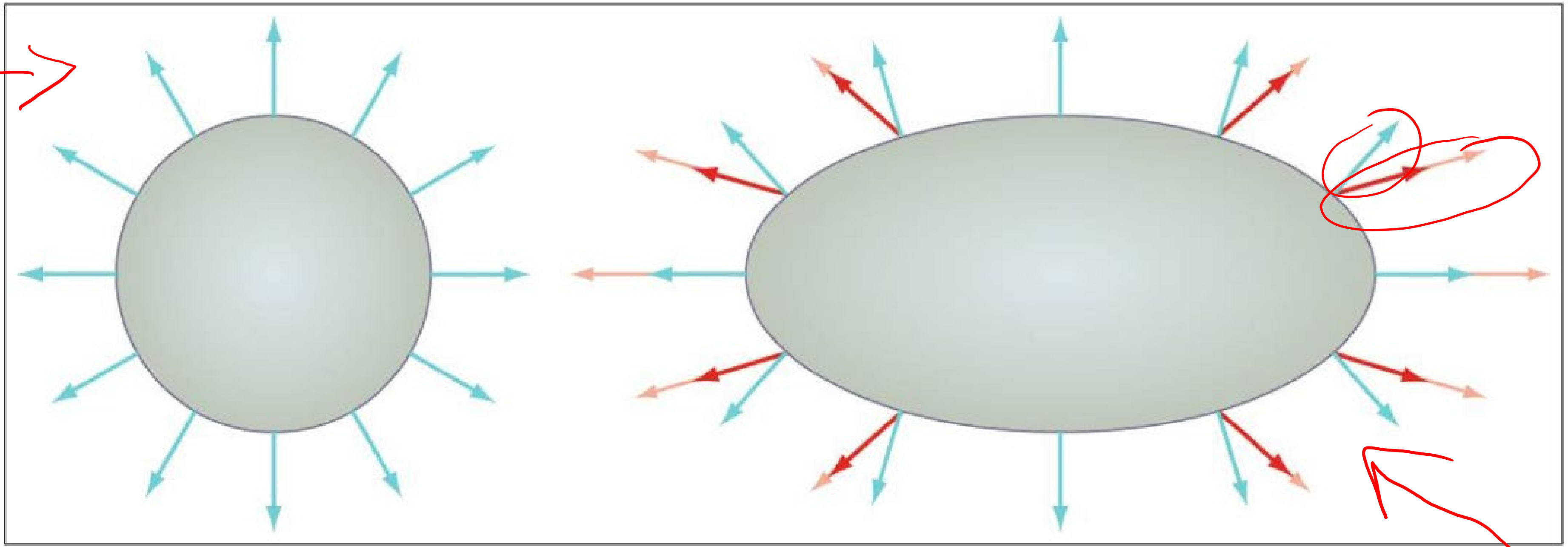


per vertex

  - Needs to be normalized

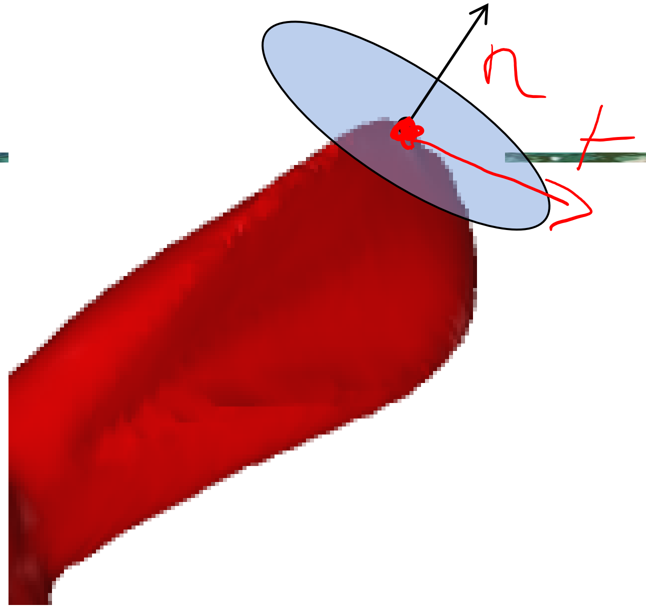
ILLINOIS

Transforming Normal Vectors



- The light red normal vectors are transformed by the same scale matrix as the vertices
- Dark red normals are unit length version of the transformed normal
- The blue normals are the correct normal…
- How can we transform the normal correctly?

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 $\mathbf{t \bullet n = t^T n = 0}$
- This should be true of the transformed geometry as well
  - Let M be the modelview matrix
- So we seek a matrix X such that
  $\mathbf{(M\,t) \bullet (Xn) = (Mt)^T (Xn) = 0}$

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

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

$t \bullet n =$

$\|t\|\,\|n\| \cos\theta$

Computing the Inverse Transpose

won't apply to n

- 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?

$$(R^T)^{-1} = R$$

$$R^T = R^{-1}$$

- 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

ILLINOIS

Inverting Affine Transformation Matrices

Recall that $AA^{-1} = I$

<u>Translation</u>

Simply translate in the opposite direction

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverting Scale Matrices

Recall that $AA^{-1} = I$

<u>Scale</u>

Simply scale by the reciprocal of the factors

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ILLINOIS

Inverting Affine Transformation Matrices

Recall that $AA^{-1} = I$

<u>Rotation</u>

The transpose is the inverse…rotates in the opposite direction

$$\begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} a & d & g & 0 \\ b & e & h & 0 \\ c & f & i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = R^T$$

Inverting Matrix Products

$$(M_1\ M_2 \ldots\ldots\ M_n)^{-1} = M_n^{-1} \ldots M_2^{-1} M_1^{-1}$$

To invert the ModelView matrix M

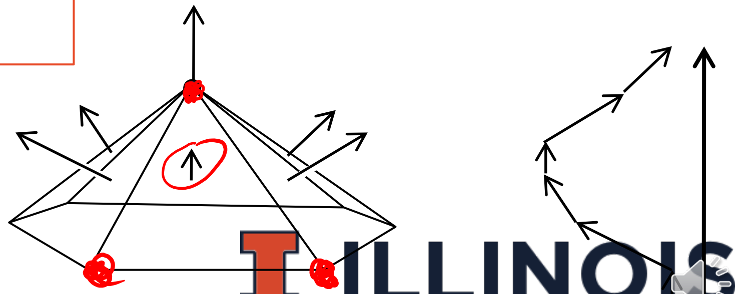
- Keep a copy of the inverse…to start both $M = I$ and $M^{-1} = I$
- For each new matrix transformation K
  $M = MK$ and $M^{-1} = K^{-1}M^{-1}$

Computing per-Vertex Normals

To compute per-vertex normal on a mesh with M vertices

- Initialize an array NArray containing M normals
  - Each normal starts as [0,0,0]
- Iterate over all triangles T=[v1,v2,v3] with v_i in CCW order
  - Compute normal N for T using N = (v2-v1)X(v3-v1)
    - NArray[v1]=(Narray[v1]+N)
    - NArray[v2]=(Narray[v2]+N)
    - NArray[v3]=(Narray[v3]+N)
- Normalize each normal in Narray to unit length

NArray[i]
normal for
vertex i



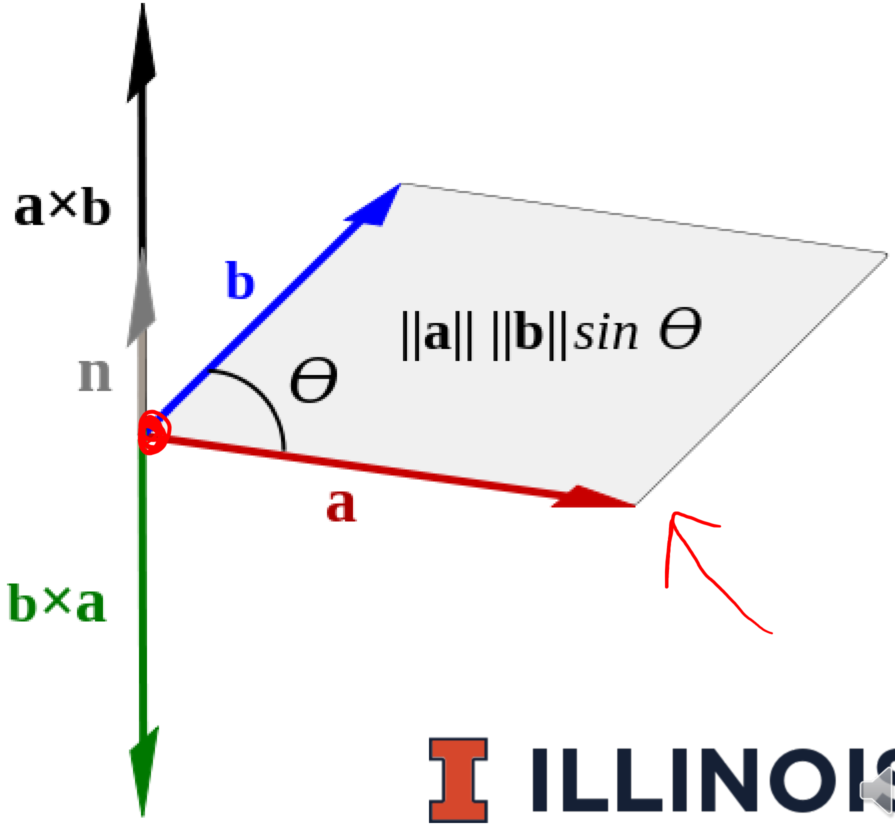
ILLINOIS

Average Vertex Normals

The previous algorithm calculates an area weighted average normal

If a and b two edges of a triangle and A is the area of the triangle

$$\underbrace{||a \times b|| = 2A}$$



ILLINOIS

Average Vertex Normals

There is no single “correct” way to calculate the average vertex normal

Assuming that the mesh is approximating a smooth surface we could:

- Use a uniformly weighted average
  - by making each triangle normal unit length before adding it in
- Use triangle area weighting
  - by scaling each triangle normal by a factor of ½
- Use parallelogram area weighting
  - by just using the normal resulting from the cross product

All of these methods are approximations and have tradeoffs

Triangle area weighting is probably most commonly used