

- The halfway vector \vec{h} is given by

$$\vec{h} = \frac{\vec{v} + \vec{\ell}}{|\vec{v} + \vec{\ell}|}$$

- Value $(\vec{h} \cdot \vec{n})$ measures deviation from ideal mirror configuration of \vec{v} and $\vec{\ell}$
- Exponent works similarly to Phong model.
- OpenGL uses Blinn-Phong model.
- Blinn-Phong better motivated physically ...
- **BUT** both models break several laws of physics (conservation of energy, for instance).
- Better physically-based and empirical models exist: Cook-Torrance, He, Ward, Lafortune's modified Phong, etc.
- Good empirical model for project:
Ashikhmin and Shirley,
An Anisotropic Phong BRDF Model,
Journal of Graphics Tools, AK Peters/ACM,
Volume 5, Number 2, 2000.

16 Shading

♠ *Shading algorithms apply lighting models to polygons, through interpolation from the vertices.*

Flat Shading: *Perform lighting calculation once, and shade entire polygon one colour.*

Gouraud Shading: *Lighting is only computed at the vertices, and the colours are interpolated across the (convex) polygon.*

Phong Shading: *A normal is specified at each vertex, and this normal is interpolated across the polygon. At each pixel, a lighting model is calculated.*

Note distinction between lighting model and shading algorithm...

16.1 Introduction

Shading

- Want to shade surfaces
- Lighting calculation for a point
Given: L_{in} , $\vec{\ell}$, and surface properties (including surface normal)
Compute: L_{out} in direction \vec{v}
- Need surface normals at every point to be lighted
- Commonly, surface is polygonal
 - True polygonal surface: use polygon normal

- Sampled surface: sample position and normal, create polygonal approximation
- Want colour for each pixel in rasterized surface

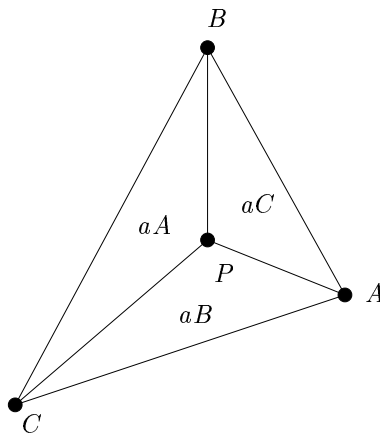
Flat Shading :

- Shade entire polygon one colour
- Perform lighting calculation at:
 - One polygon vertex
 - Center of polygon
 - What normal do we use?
 - All polygon vertices and average colours
- Problem: Surface looks faceted
- OK if really is a polygonal model, not good if a sampled approximation to a curved surface.

♠ Readings: Watt: 6.3. Red book: 14.2. White book: 16.2.4, 16.2.5. Hearn and Baker: 14.5.

16.2 Gouraud Shading

- **Gouraud shading** interpolates colours across a polygon from the vertices.
- Lighting calculations are only performed at the vertices.
- Interpolation well-defined for triangles.
- Extensions to convex polygons ... but not a good idea, convert to triangles.
- Barycentric combinations are also **affine combinations**...
Triangular Gouraud shading is **invariant** under affine transformations.



$$aA = \Delta PBC / \Delta ABC$$

$$aB = \Delta APC / \Delta ABC$$

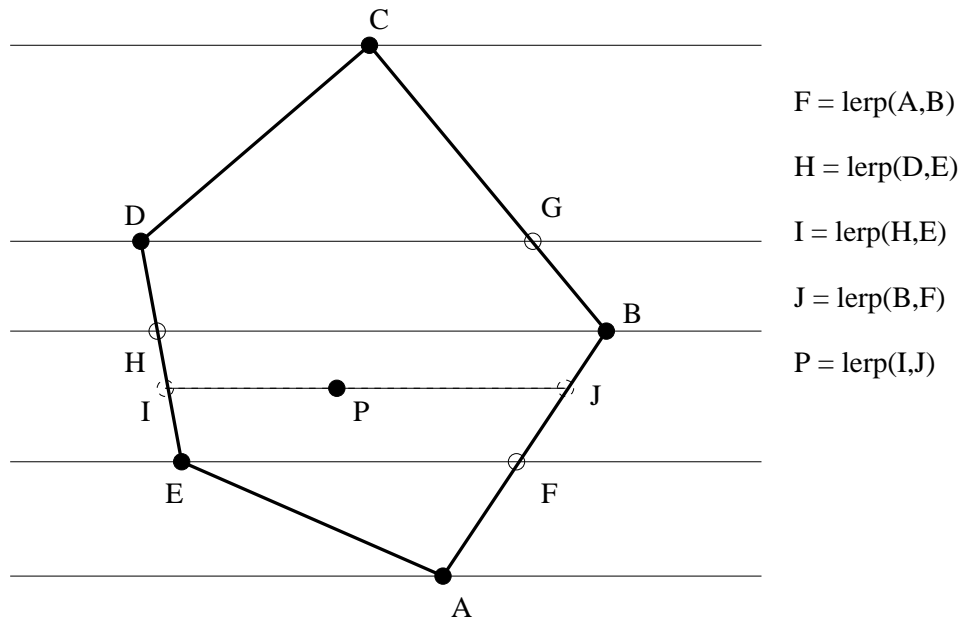
$$aC = \Delta ABP / \Delta ABC$$

$$aA + aB + aC = 1$$

$$P = aAA + aBB + aCC$$

- To implement, can use repeated affine combination along edges, across spans, during rasterization.
- Gouraud shading is well-defined only for triangles

- For polygons with more than three vertices:
 - Sort the vertices by y coordinate.
 - Slice the polygon into trapezoids with parallel top and bottom.
 - Interpolate colours along each edge of the trapezoid. . .
 - Interpolate colours along each scanline.



- Gouraud shading gives **bilinear** interpolation within each trapezoid.
- Since rotating the polygon can result in a different trapezoidal decomposition, n -sided Gouraud interpolation is **not affine invariant**.
- Aliasing also a problem: highlights can be missed or blurred.
Not good for shiny surfaces unless fine polygons are used.
- Exercise: Provide an example of the above effect.
- Exercise: Prove repeated affine combinations (the above algorithm) is equivalent to barycentric combinations on triangles.
- Exercise: Prove that the extension of the repeated affine combination algorithm to arbitrary polygons is not invariant under affine transformations.
- Common in hardware renderers; the model that (classic) OpenGL supports.
- Linear interpolation in device space not consistent with linear interpolation in world space. Modern implementations of OpenGL actually implement rational linear interpolation, which takes perspective into account.

16.3 Phong Shading

- **Phong Shading** interpolates lighting model parameters, **not** colours.
- Much better rendition of highlights.
- A **normal** is specified at each vertex of a polygon.
- Vertex normals are independent of the polygon normal.
- Vertex normals should relate to the surface being approximated by the polygonal mesh.
- The normal is interpolated across the polygon (using Gouraud techniques).
- At each pixel,
 - Interpolate the normal. . .
 - Interpolate other shading parameters. . .
 - Compute the view and light vectors. . .
 - Evaluate the lighting model.
- The lighting model does not have to be the Phong lighting model!
- Normal interpolation is nominally done by vector addition and renormalization.
- Several “fast” approximations are possible.
- The view and light vectors may also be interpolated or approximated.
- Problems with Phong shading:
 - Distances change under perspective transformation
 - Where do we do interpolation?
 - Normals don’t map through perspective transformation
 - Can’t perform lighting calculation or linear interpolation in device space
 - Have to perform lighting calculation in *world space* or *view space*, assuming model-view transformation is affine.
 - Have to perform linear interpolation in world or view space, project into device space
 - Results in rational-linear interpolation in device space!
 - Interpolate homogenous coordinates, do per-pixel divide.
 - Can be organized so only need *one* division per pixel, regardless of the number of parameters to be interpolated.
- Phong shading, and many other kinds of advanced shading, can be simulated with programmable vertex and fragment shaders on modern graphics hardware:
 - Classic Gouraud shading is linear in device space.
 - Modern graphics hardware performs rational linear interpolation — looks like interpolation happens in world space.

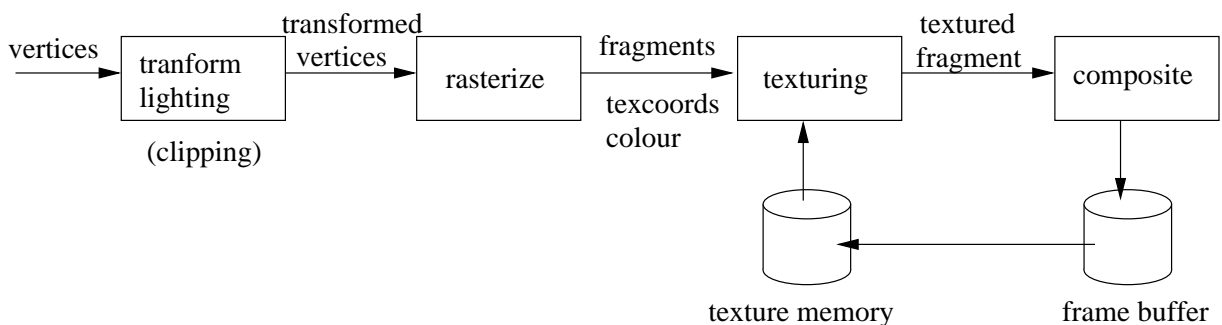
- Interpolate normals, view vectors, and light vectors using generic interpolation hardware
- Usually also interpolate diffuse term after computing it in vertex shader.
- Write fragment shader to renormalize normals to unit length, compute lighting model, and assign colour to pixel.
- Can also use texture maps as lookup tables. . .

17 Graphics Hardware

17.1 Graphics Hardware

- Graphics hardware has been evolving over the past several decades
- In the 60's, 70's, vector displays costing millions of dollars
US military was main (only) customer
- Raster displays started appearing in the 70's
- In the 80's, raster displays and graphics hardware cost 10's to 100's of thousands of dollars.
Cost low enough that companies could afford the hardware.
SGI won the battle of work station graphics, OpenGL became standard
- In the 90's, graphics hardware started moving to PC's
DirectX is MicroSoft's challenge to OpenGL
- In the 00's, PC graphics hardware has displaced work station graphics
Good, inexpensive cards for PCs
- Currently getting faster in excess of Moore's law
- Many graphics cards are "more powerful" than CPU

Traditional OpenGL (1.2)



- A fragment is a sort of virtual pixel
- Clipping done in texture lighting portion