Computação Gráfica

Computer Graphics

Engenharia Informática (11569) – 3° ano, 2° semestre

Chap. 9 – Shading

http://di.ubi.pt/~agomes/cg/09-shading.pdf

Outline

Based on
https://www.cs.unc.edu/~dm/UNC/COMP236/LECTURES/LightingAndShading.ppt

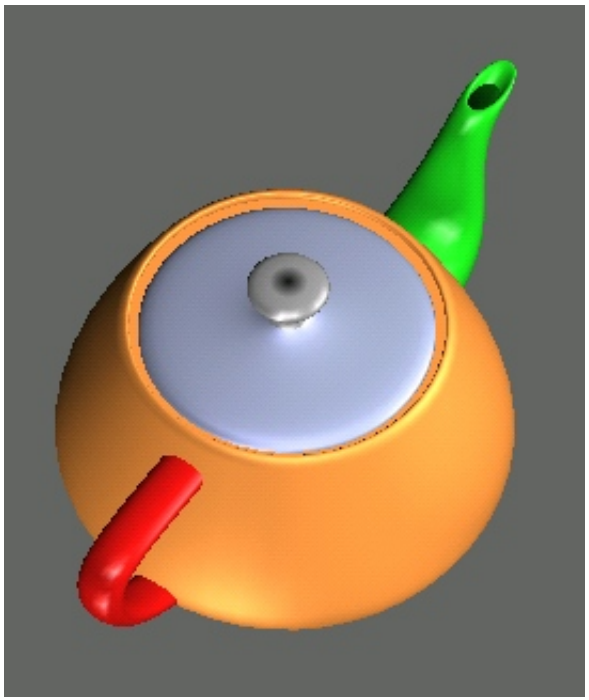
…:

- Light-dependent illumination models: a refresher
- Shading: motivation
- Types of shading: flat, Gouraud, and Phong
- Flat shading algorithm
- Gouraud shading algorithm
- Phong shading algorithm
- Shading issues
- Flat, Gouraud, and Phong shaders in GLSL
- OpenGL/GLSL examples.

Light-dependent models: a refresher

Direct (or local) illumination:

- Simple interaction between light and objects
- Real-time process supported by OpenGL
- Example: *Phong's illumination model*.



http://jedi.ks.uiuc.edu/~johns/raytracer/raygallery/stills.htm

Indirect (or global) illumination:

- Multiple interactions between light and objects (e.g., inter-object reflections, refraction, and shadows)
- It is a real-time process for small scenes, but not for complex scenes
- Examples: *raytracing, radiosity, photon mapping …*

Shading: motivation

We now have a *direct lighting model* for a simple point on the surface.

Assuming that our surface is defined as a mesh of polygonal faces, what points should we use?

- Computing the color for all points is expensive
- The normals may not be explicitly defined for all points

It should be noted that:

- Lighting involves a rather cumbersome calculation process if it is applied to all points on the surface of an object
- There are *several possible solutions*, each of which has different implications for the visual quality of the scene.

Shading

Types of shading

In polygonal rendering, there are 3 main types:

- Flat shading
- Gouraud shading
- Phong shading

These roughly correspond to:

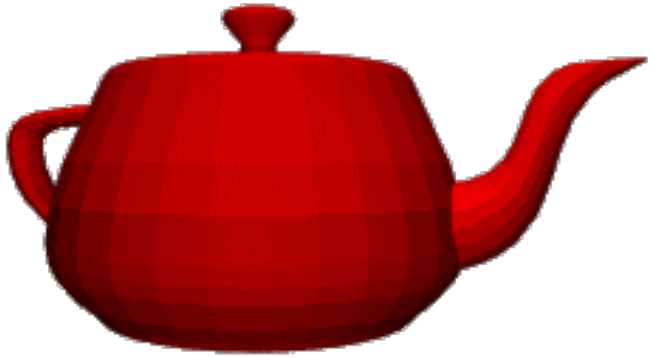
- Per-polygon shading
- Per-vertex shading
- Per-pixel shading

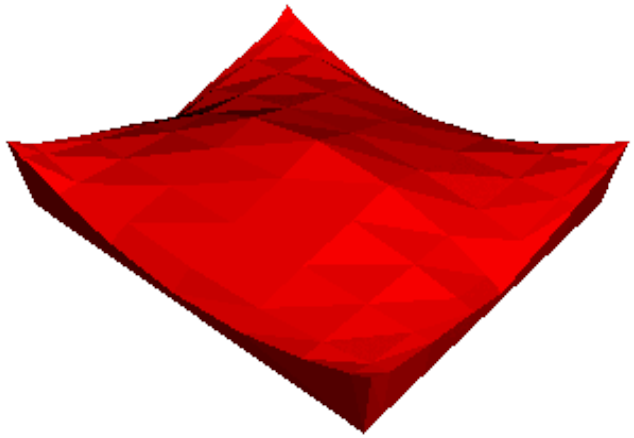
Recall that new graphics hardware makes programmable shading per pixel!

Flat shading

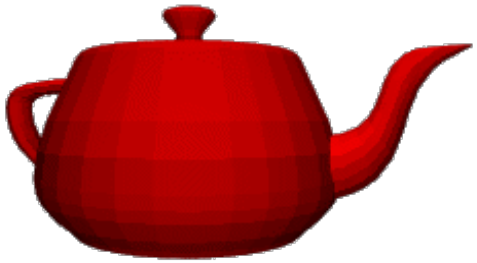
Flat shading:

- It is the fastest and simplest method,
  - because it computes the color (or shade) at a single point of each polygon,
  - and use that color on every pixel of the polygon
- The lighting intensity (color) is the same for all points of each polygon.
- Benefit:
  - Fast: a shade per polygon
  - because it uses a normal per polygon
- Disadvantages:
  - Inaccurate
  - Discontinuities at the boundaries of polygons
  - Lack of realism





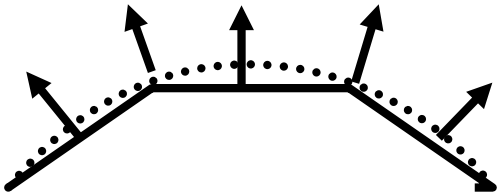
Flat shading: lack of realism



Lack of realism results from:
- *The use of faceted objects (or meshes)*
- *The use of a single normal per polygon*

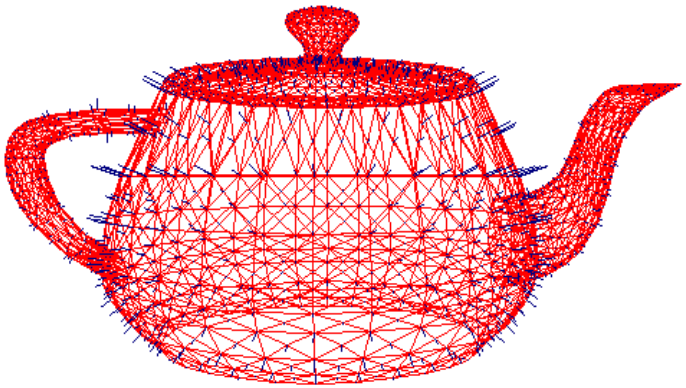
Why?

- For point light sources, the direction to the light source varies for each point of the facet.
- In the case of specular reflection, the direction to the observer varies for each point of the facet.

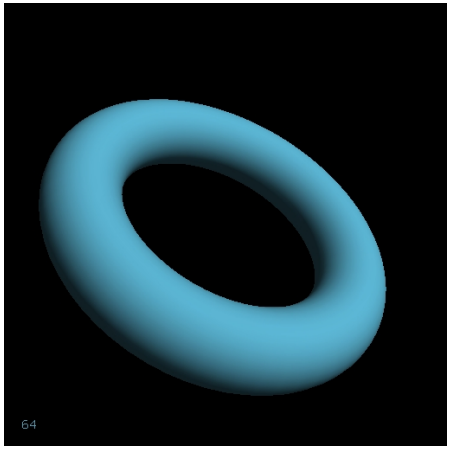


Solutions:

- To obtain visually smooth surfaces, we must use normal vectors at the vertices of the polygons
  - normally different from polygon normal
  - are used only for shading effects
  - It suggests a better approximation to the real surface that the polygons approach
- The vertex normals can be
  - supplied with the model
  - approximated by the average of the normals to the facets that share each vertex



Gouraud shading: algorithm



- It illuminates or directly shades each vertex using its position and its normal vector.
- Interpolate linearly the color on each face: first along the edges, then along scanline lines inside it.

Compute S_A, S_B, S_C for triangle ABC.

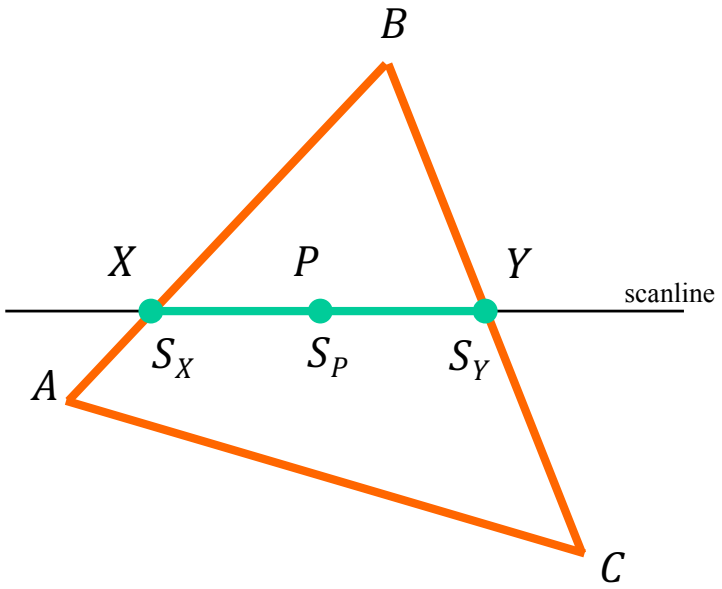
- S_i = shade of point i.

For a scanline XY, compute S_X, S_Y by lerping.

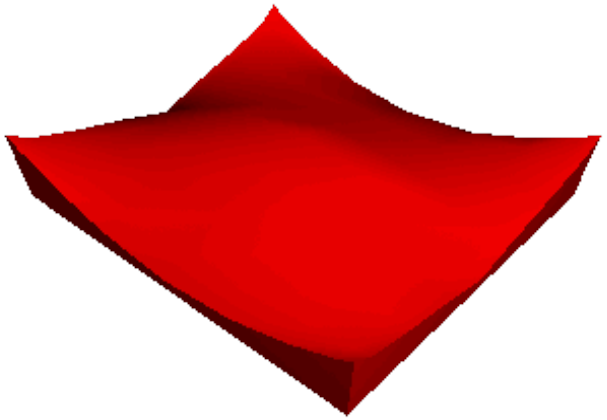
- $t_{AB} = |AX| \,/\, |AB|$
- $S_X = t_{AB}\, S_A + (1 - t_{AB}) S_B$

Compute S_P

- By lerping between S_X and S_Y.



Gouraud shading: issues



Benefits:

- Fast: incremental calculations during rasterization
- Smoother: a normal vector is used for each shared vertex in order to obtain continuity between faces

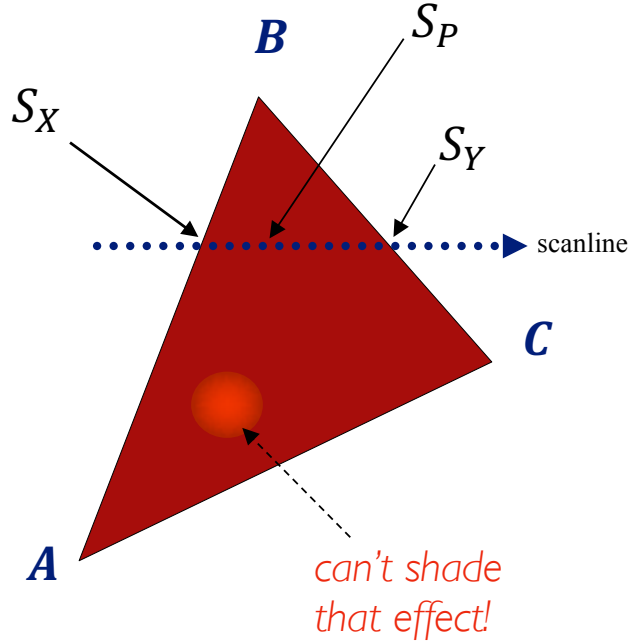
Disadvantages:

- Still inaccurate. The polygons appear bumpy and dull.
- It tends to eliminate the specular component of light. If we include it, it will be distributed over the entire polygon.

Issues:

- Perspectively-incorrect interpolation
- Mach banding

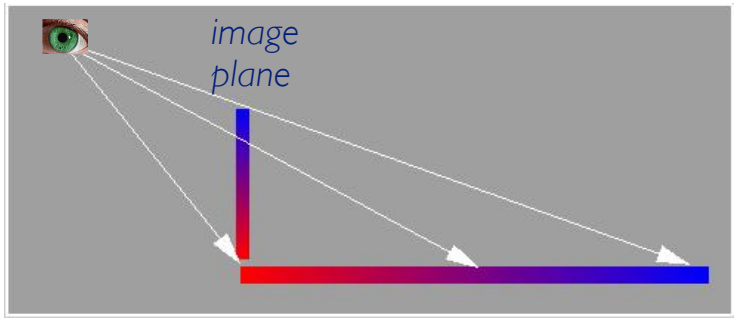
$$S_X = S_A + t_1(S_B - S_A)$$
$$S_Y = S_B + t_2(S_C - S_B)$$
$$S_P = S_X + t_3(S_Y - S_X)$$



Perspective distortion

Perspective distortion

- Perspective projection complicates linear interpolation, resulting in *perspectively-incorrect interpolation*.
- Linear interpolation in the screen space is not aligned with linear interpolation in the space of the scene domain. Relationship between screen space distance and eye space distance is nonlinear.
- Therefore, relationship between interpolation in the two spaces is also nonlinear.
- Thus, screen space linear interpolation of colors (and texture coordinates) results in incorrect values



Z – into the scene

Possible solution

- Larger polygons are partitioned into smaller polygons to reduce distortion.

Perspectively-correct interpolation

- Can we interpolate in eye space, then project every interpolated point?: *way too much work*!
- Can we interpolate in screen space and correct for perspective nonlinearity?: *yes*!

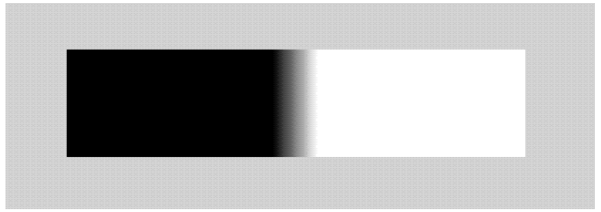
For a detailed derivation, see:

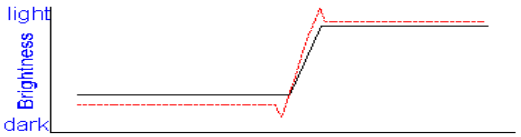
- https://www.comp.nus.edu.sg/~lowkl/publications/lowk_persp_interp_techrep.pdf
- https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/perspective-correct-interpolation-vertex-attributes

Here, we skip to the punch line:

- Given two eye space points, E_1 and E_2.
  - Can lerp in eye space: $E(T) = E_1(1-T) + E_2(T)$.
  - T is eye space parameter, t is screen space parameter.
- To see relationship, express in terms of screen space t:
  - $E(t) = [\,(E_1/Z_1).(1-t) + (E_2/Z_2).t\,] \,/\, [\,(1/Z_1).(1-t) + (1/Z_2).t\,]$
  - E_1/Z_1, E_2/Z_2 are projected points.
  - Because Z_1, Z_2 are depths corresponding to E_1, E_2.
- Looking closely, can see that interpolation along an eye space edge = interpolation along projected edge in screen space divided by the interpolation of $1/Z$.

Mach bands





http://www.markschenk.com/various/machband.html

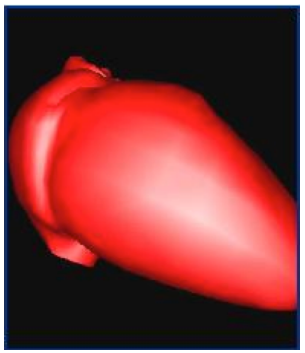
Gouraud discusses "artifact" of lerping.

Mach bands:

- Describe the increase of contrast of adjacent colors
- Caused by interaction of neighboring retinal neurons.
- Acts as a sort of high-pass filter, accentuating discontinuities in first derivative.
- Linear interpolation causes first derivative discontinuities at polygon edges

banded along edges

flat shading

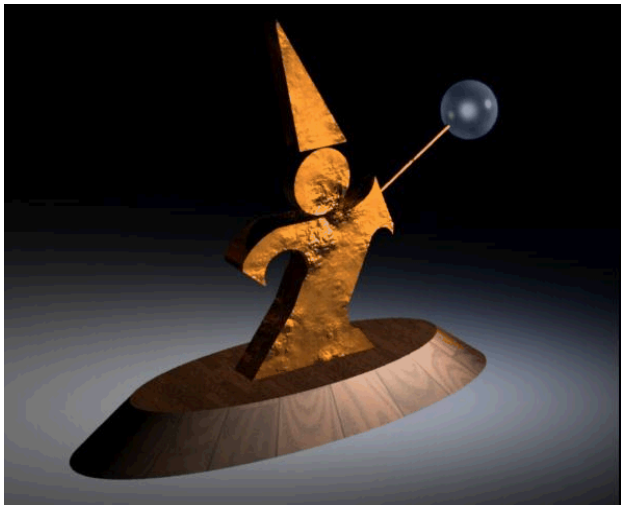


Gouraud shading

Gouraud suggests higher-order interpolation would alleviate mach banding.

- But stresses the performance cost.
- Probably not worth it.

Phong shading helps the problem.



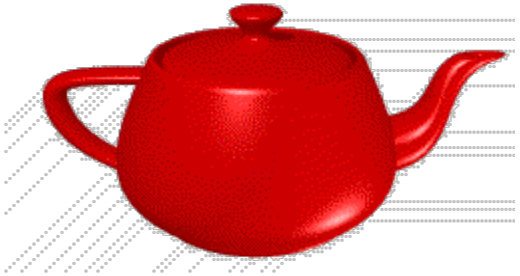
floor appears banded

Phong shading

- Phong shading is not the same as Phong lighting, although they are concepts sometimes confused with one another.
- Phong lighting: the empirical model we discussed in the previous chapter to calculate the illumination at a point on the surface of an object.
- Phong shading: linearly interpolates the normal in any and every facet by applying the Phong illumination model to each pixel

Earlier graphics hardware did not implemented Phong shading

- APIs (D3D, OGL) employ Blinn-Phong *lighting* and Gouraud *shading*.

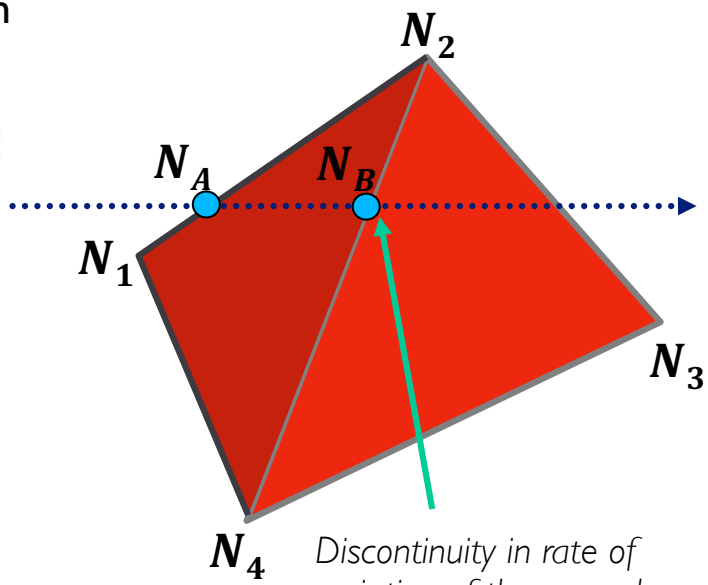


Phong shading applies lighting computation *per-pixel*

- Uses linear interpolation of normal vectors, rather than colors.

Interpolation just as with colors in Gouraud shading

- Interpolate scan line endpoint normals N_A, N_B from endpoints of intercepted edges.
- Interpolate normal N_P at each pixel from N_A, N_B.
- Normalize N_P.
  - (Interpolation of unit vectors does not preserve length).
- Back-transform N_P to eye space, compute lighting.



Discontinuity in rate of variation of the normal it's harder to detect

Phong shading: improvements and issues

$$I = I_E + k_A I_A + \sum_{i=1}^{\#lights} k_D I_{D_i}(\boldsymbol{n} \bullet \boldsymbol{l_i}) + k_S I_{S_i}(\boldsymbol{v} \bullet \boldsymbol{r_i})^h$$

Interpolates linearly from normals to vertices

- Computes the lighting equations in pixel
- You can use the specular component
- It should be noted that the normals are used to compute the diffuse and specular components

Results are much improved over Gouraud.

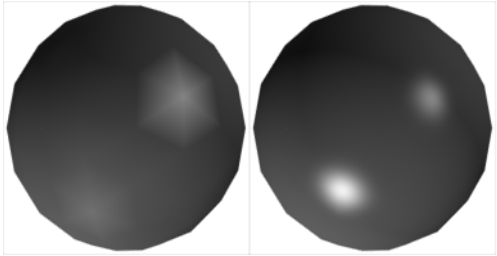
- Harder to tell low- from high-polygon models.

Still some issues:

- Still Mach banding because can still get first derivative discontinuities
- Polygonal silhouettes still present, particularly in low-resolution tessellations
- Shared vs. unshared vertices.
- Perspective distortion
- Interpolation depending on the orientation of the polygons
- Shared vertex problems
- Erroneous mean at vertices

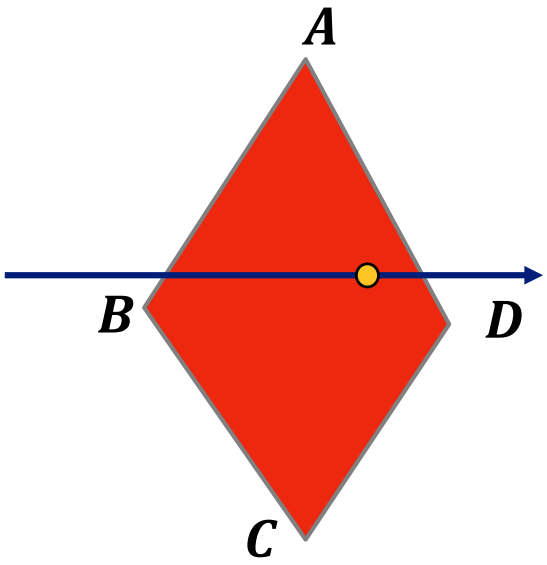
Phong shading issues: polygonal silhouettes and orientation of polygons

Polygonal silhouettes



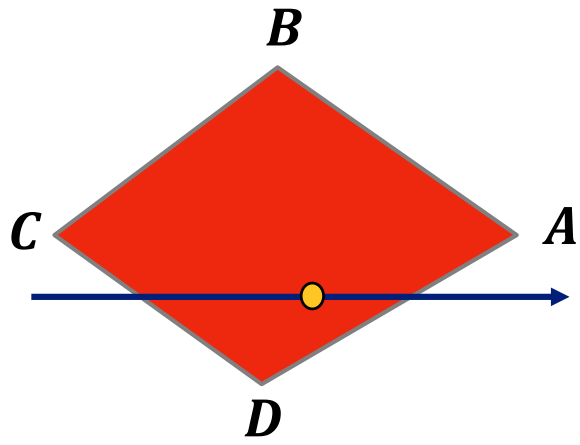
Gouraud *Phong*

Interpolation depending on the orientation of the polygons



Interpola entre AB e AD

Rotate -90º and shades The same point



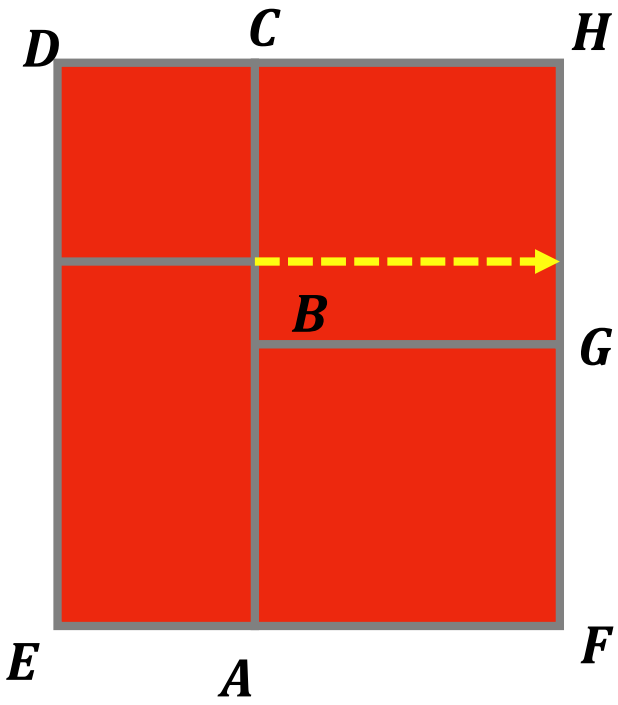
Interpola entre CD e AD

Phong shading issues: shared vertices

Shared vertex problems

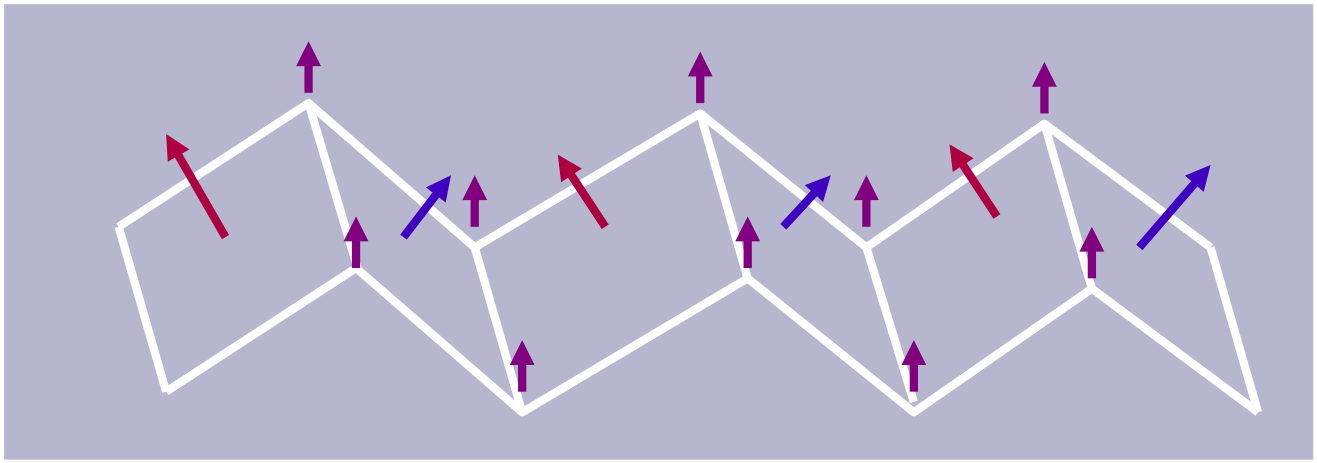
Example:

- *The vertex B is shared by two rectangles on the right, but not by the one on the left*
- *The first segment of the scanline (in yellow) is interpolated between DE and AC*
- *The second segment of the scanline is interpolated between BC and GH*
- *A discontinuity may arise*



Phong shading issues: erroneous mean of normals

Erroneous mean of normals at vertices



Shading by direct illumination: summing up

Summary:

- *We only superficially approached surface illumination. The common model is clearly unsuitable for accurate lighting, but has the advantage of being fast and simple.*
- *It takes into account two sub-lighting problems:*
    - *Where does the light go? Light Transport*
    - *What happens on surfaces? Light Reflection*

Flat Shading

- Per-polygon shading
- Phong lighting is computed once for each polygon

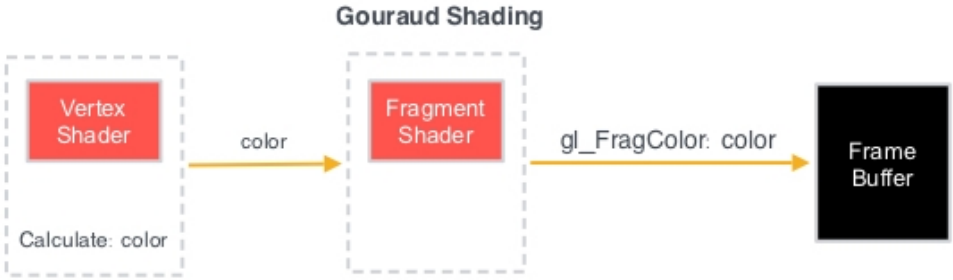
Gouraud Shading

- Per-vertex shading
- The Phong illumination is computed at the vertices and interpolates the shade values (or colors) over the polygon

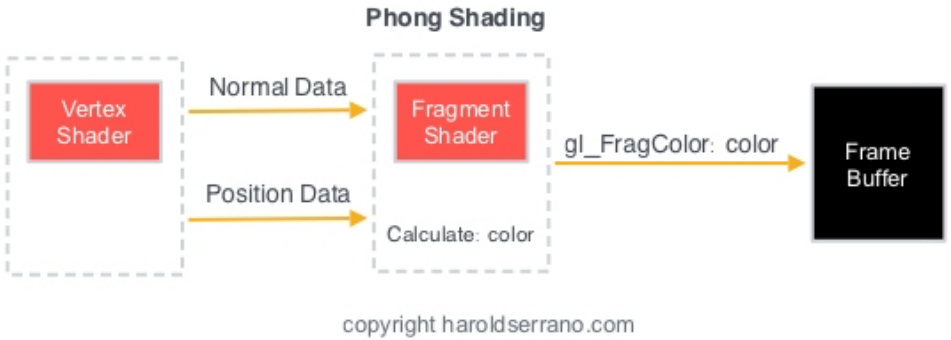
Phong Shading

- Per-pixel shading
- Compute the mean of the normals of the faces incident on each vertex
- The normals are interpolated over the face calculating the Phong illumination at each point tied to a pixel.

Gouraud and Phong shaders



Gouraud shading *is a per-vertex color computation. What this means is that the vertex shader must determine a color for each vertex and pass the color as an out variable to the fragment shader. Since this color is passed to the fragment shader as an in variable, it is interpolated across the fragments thus giving the smooth shading.*



In contrast, ***Phong shading*** *is a per-fragment color computation. The vertex shader provides the normal and position data as out variables to the fragment shader. The fragment shader then interpolates these variables and computes the color.*

Examples in OpenGL

See examples in Chapter 2 and 3 of the book entitled "***OpenGL 4 Shading Language Cookbook***" (2nd edition)
Authored by David Wolff; these examples were run in both theoretical and lab classes, and can be found at:
https://github.com/daw42/glslcookbook

Other types of per-pixel shading

Ray tracing

- Doesn't use Gouraud or Phong shading.
- Each pixel uses own ray to determine color.
  - Can apply arbitrary lighting model.
  - Classical (Whitted) ray tracing uses Phong model.
- Since ray tracing determines colors based on intersections, don't have to use polygonal geometry.
  - Thus, can potentially use exact normals, rather than interpolation.

New hardware provides per-pixel capabilities

- e.g. NVIDIA pixel shaders.
- Allow (somewhat) arbitrary programs on each pixel.
- So new hardware *can* implement Phong shading.

Also, vertex programs

- Allow (somewhat) arbitrary programs on each vertex.

References

Gouraud, Phong, Blinn papers I handed out.

- Available in *Seminal Graphics*, ACM press.

Glassner, *Principles of Digital Image Synthesis*, volume two.

- Highly detailed and low level.

Möller and Haines, *Real-Time Rendering*.

- A great book, with the best bibliography you can find.

Rogers, *Procedural Elements for Computer Graphics*.

- One of my favorites.

Foley, van dam, *et al*. *Computer Graphics, Principles and Practice*.

- Not the best treatment, but it covers everything.

Summary:

…:

- Light-dependent illumination models: a refresher
- Shading: motivation
- Types of shading: flat, Gouraud, and Phong
- Flat shading algorithm
- Gouraud shading algorithm
- Phong shading algorithm
- Shading issues
- Flat, Gouraud, and Phong shaders in GLSL
- OpenGL/GLSL examples.