Graphics rasterization pipeline

Light reflection model
Conversion to screen coordinates
Perspective transform
Scan-conversion
Performance issues
Compare with raycasting

Rasterization (high level)

For each triangle T do

- Compute vertex colors (or triangle color)
- Clip to viewing frustum
- Compute perspective projection T'
  - by projecting its vertices
- Interpolate vertex depth and color
  - For each pixel p' in T'
    - If (interpolated depth < stored depth)
      - Update color and depth of p'

Graphic Pipeline

Model transforms

Vertex lighting

Viewing transform

Frustum clipping

Perspective transform

Slope computation

Scan-conversion

Approximate light reflection model

Reflected light: $k_a + \left(k_s(V \bullet R)^n + k_d(N \bullet L)\right) I/r^2$

Ambient light

Ambient reflection coefficient: k_a

Distance attenuation of incident light: I/r^2

I = Intensity of point source at distance d in direction L

The attenuation effect is often softened to $\max(I/(ar^2 + br + c), I)$

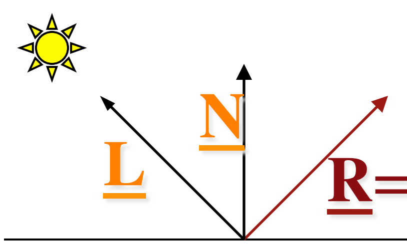
Diffuse (Lambertian) reflection: $k_d(N \bullet L)$

k_d = diffuse reflection coefficient

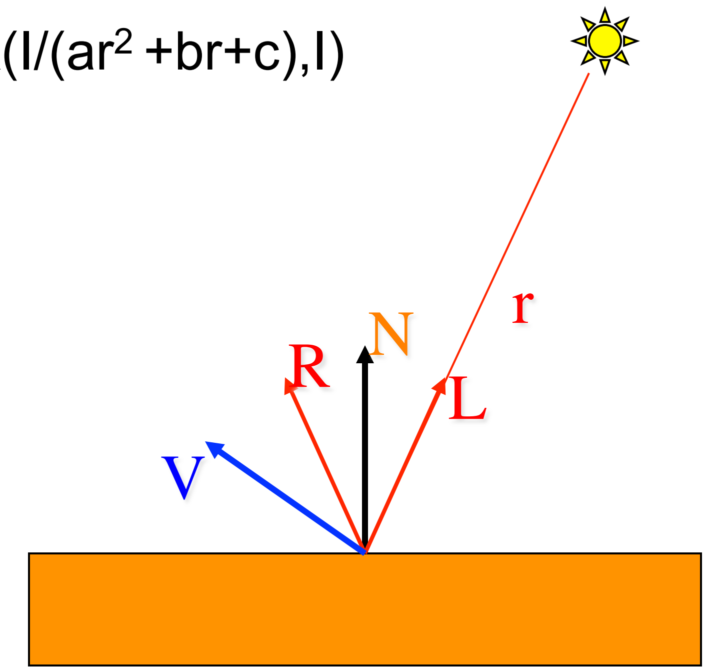
Specular (Phong) reflection: $k_s(V \bullet R)^n$

k_s = specular reflection coefficient

n = specular reflection exponent



$\underline{\mathbf{R}} = 2(\underline{\mathbf{N}} \bullet \underline{\mathbf{L}})\underline{\mathbf{N}} - \underline{\mathbf{L}}$



Graphic Pipeline

Model transforms

Vertex lighting

Viewing transform

Frustum clipping

Perspective transform

Slope computation

Scan-conversion

Clipping

What does it do?

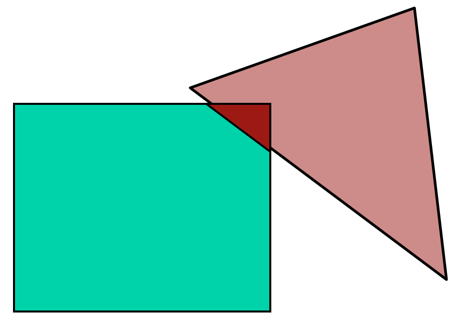
- Trims each triangle to the window

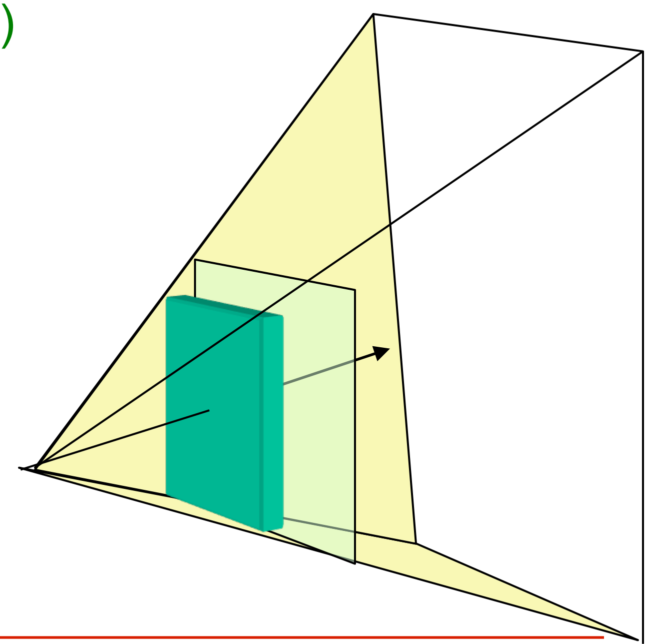
Why?

- To avoid processing out of window pixels

How?

- Preprocessing stage
  - In model space (before perspective transform)
    - Intersect the triangle the viewing frustum
      - Intersection of 5 or 6 half-spaces
  - Or in image space (after perspective)
    - Using axis-aligned half-spaces





Graphic Pipeline

Georgia Tech College of Computing
Interactive & Intelligent Computing

Model transforms

Vertex lighting

Viewing transform

Frustum clipping

Perspective transform

Slope computation

Scan-conversion

Perspective transform

Screen center O, is the closest point on the screen to the viewpoint V

Given S on screen and the screen normal K, O=V+dK, with OS•K=0. Hence, OV•K+dK•K=0 and d= –OV•K.

Compute the (x,y,z) screen coordinates of P

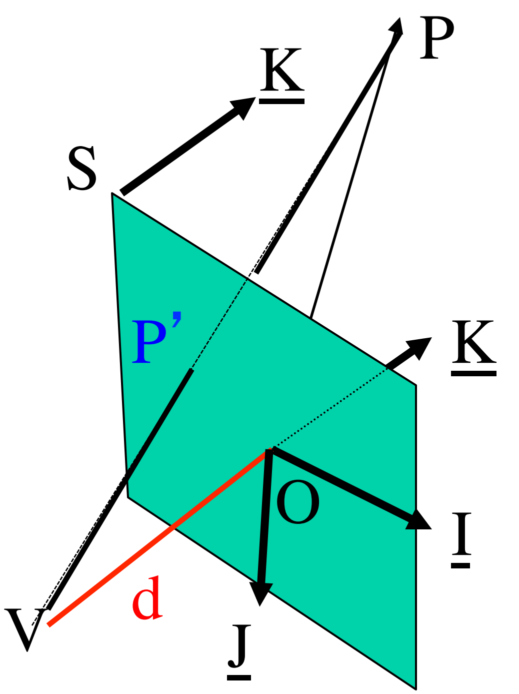
x=<u>OP</u>•<u>I</u>, y=<u>OP</u>•<u>J</u>, z=<u>OP</u>•<u>K</u>

Compute perspective P' of P

P' = (x',y',z') = d/(d+z) P

P appears on screen at (x',y')

z' is stored at pixel containing (x',y')



Graphic Pipeline

Georgia Tech College of Computing
Interactive & Intelligent Computing

Model transforms

Vertex lighting

Viewing transform

Frustum clipping

Perspective transform

Slope computation

Scan-conversion

Scan-convesion (raserization)

Clear **z-buffer Z[*,*] and back frame buffer I[*,*]**

For each triangle **(A,B,C) do**

Lit **A, B, C using normals, surface attributes, lighting parameters**

Transform **vertices {A'=dA/(d+A.z); B'=dC/(d+B.z); C'=dC/(d+C.z)}**

Compute slopes **for c (color), z (depth), and leading/trailing edges**

For each scanline **L covered by triangle (A',B',C') do**

Compute range [S..E] by interpolation on leading/trailing edges

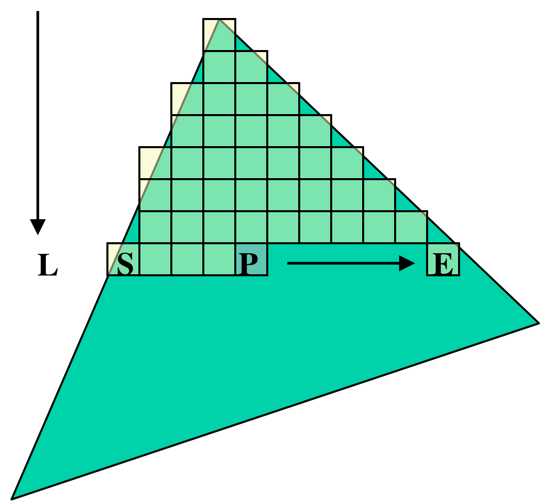
Compute initial z and c by interpolation along leading edge

For each pixel **P in [S..E] do**

Compute z and c using interpolation

If z < Z[P,L] then {Z[P,L]:=z; I[P,L]:=c}

Swap **frame buffers to show image**



Rasterization costs

Per vertex

- Lighting calculations
- Perspective transform
- Slopes calculation

Per pixel

- Interpolations of z, c
  - **Along leading edges (per triangle)**
  - **Along scanline (per pixel)**
- Read, compare, right from/to (graphics) memory
    - Read is sequential along scanline
  - z-buffer (read and write)
  - Image buffer (write)

Ray-casting / scan-conversion

Scan-conversion:

For each triangle T do

For each pixel P **covered by the projection of T do**

Decide if the triangle is visible

Compute reflected color and store in the pixel

Ray-casting:

For each pixel P do

For each triangle T **projecting over P** do

Decide if the triangle is visible

Compute reflected color and store in the pixel

Why is scan-conversion faster

Ray casting consider all ray-triangle pairs.

Scan-conversion considers only pixels in projections

10,000 times less work if you have 10x10 pixel triangles

Ray-casting requires multiplications and divisions

for selecting which triangles cover a pixel and for computing the corresponding depth.

Scan-conversion uses mostly additions

to test which pixels are covered and to compute the color and depth for each pixel

And a few multiplications and division per triangle

to compute vertex projections and slopes),