## Rendering Systems Issues

**Design goals**

- **Complexity**
- **Diversity**
- **Fast**
- **High fidelity; no computational artifacts**

**Interfaces**

- **Geometric shapes**
- **Visibility subsystem**
- **Shading subsystem**
- **Window and 2D graphics systems**



## Geometric Primitives

**Modeling** *Houses, Chairs, Arches*

↓

**Rendering** *Fractals, Spheres, Patches, Polygons*

↓

**Intermediate** *Micropolygons, Blots, Spans, Samples*

↓

**Display** *Pixels*

## Geometric Methods

**Simple object-oriented architectures**

- **Bounds (culling)**
- **Transform (modeling and viewing)**
- **Surface normals (shading)**
- **Surface parameterization (texturing)**
- **Rasterization/intersection (visibility)**
- **Conversion**
- **Miscellanous: create, load, save, ...**



## Hidden/Visible Surface Algorithms

**Variations**

- **Object space vs. image space**
  ***Continuous vs. discrete***
- **Object order vs. image order**
  ***Forward vs. backward***
- **Sorting time**
  ***Sort first vs. sort last***

# Object Space Algorithms

**Robert's Algorithm**

```
for each polygon
    construct shadow volume
for each edge
    clip edge to shadow volumes
```

**Weiler-Atherton Algorithm**

```
sort polygons front to back
S = NULL
for each polygon P
    P = Difference(P,S)
    S = Union(P,S)
```



# Sort-first vs. Sort-last

**Painter's Algorithm (Schumacher '69)**

```
sort polygons back to front
for each polygon P
    for each pixel S in P
        image[S.x,S.y].c = S.c
```

**Z-buffer Algorithm (Catmull '74)**

```
for each polygon P
    for each pixel S in P
        if S.z < image[S.x,S.y].z
              image[S.x,S.y].z = S.z
              image[S.x,S.y].c = S.c
```

# Object vs. Image Order

**Ray Casting (Appel '68, Goldstein&Nagel '71)**

```
for each pixel S in the image
   for each polygon P
       zmin = +huge
       if polygon.inside(S.x,S.y,&z,&c)
          if z < zmin then S.c = c
```

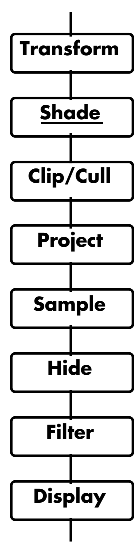
**Z-buffer Algorithm (Catmull '74)**

```
for each polygon P
   for each pixel S in P
       if S.z < image[S.x,S.y].z
              image[S.x,S.y].z = S.z
              image[S.x,S.y].c = S.c
```

***SSS: "ZB and RT algorithms are hopelessly inefficient"***



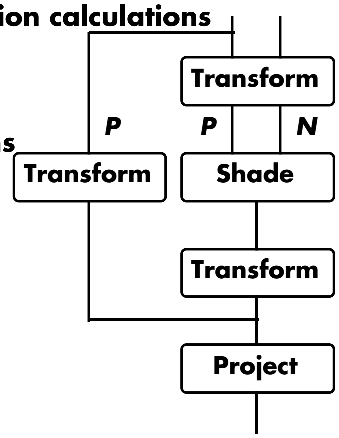
# Current Graphics Pipeline



## Lighting

**Illumination and reflection calculations**

- **Normals**
- **Coordinate systems**
  **Lighting calculations are derived from laws of physics must be done in cs isometric to world**



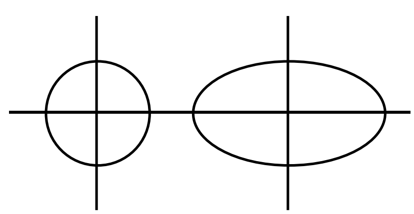



## Transformation of Normals

**Normals remain perpendicular to tangents**

$$N \bullet V = 0$$

$$N' \bullet V' = 0$$



**Rotations special case!**

# Shading

***Efficient display of curved surfaces***

**Interpolation**

- **Flat shading**

  ***Fill interior with constant color***

  **Determine polygon color by lighting calculation**

```
bgnpolygon;
   c3f;
   v3f; v3f; v3f;
endpolygon;
```

- **Gouraud shading**
- **Phong shading**



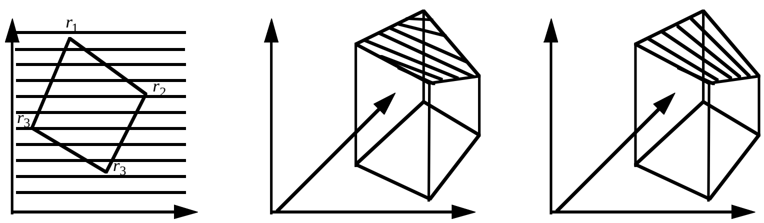
# Gouraud Shading

**Interpolate**

- **Linearly in screen space to fill interior**
- **Interpolate color, depth, texture, ...**

```
bgnpolygon;
 c3f;v3f;
 c3f;v3f;
 c3f;v3f;
endpolygon;
```

## Gouraud Shading

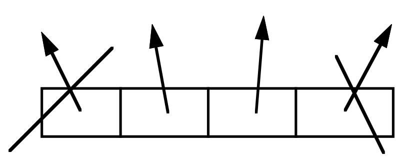





## Phong Shading

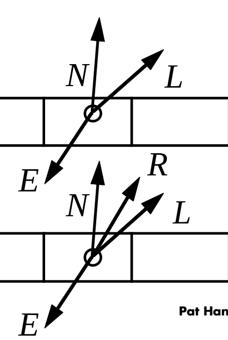
**Three Ideas!**

**1. Interpolate normals**



**2. Compute lighting once per pixel**

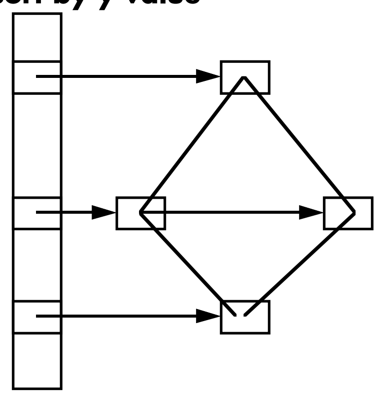
**3. Phong reflection formula**




$$(R \bullet L)^s$$

## Scanline Algorithms

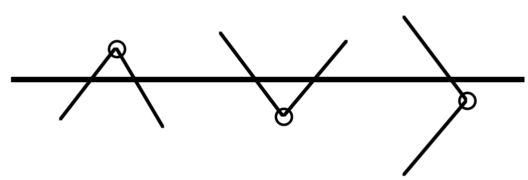
1. For each polygon vertex
   Bucket sort by y value



CS348B Lecture 2 Pat Hanrahan

## Scanline Algorithms

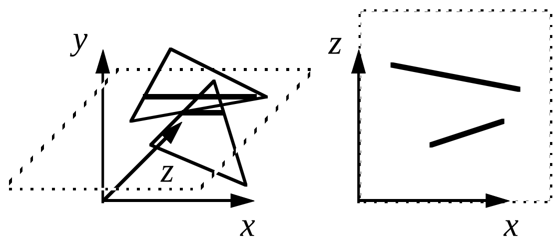
2. For each scan-line maintain active edge list
   For each vertex in this scan-line bucket
      if upper vertex then insert 2 edges
      if lower vertex then delete 2 edges
      if monotone vertex then replace edge
   Sort active edge list by x



CS348B Lecture 2 Pat Hanrahan

## Scanline Algorithms

**Reduce 3D hidden surface to 2D hidden edge**






## 2D Hidden Edge Algorithms

1. **Scan-line z-buffer**
2. **Painter's algorithm**

```
Z sort spans
for each span
  paint span
```

3. **X-sort**

```
X sort spans
for x=xmin to xmax
  if edge at x
     if left then insert span, z-sort
     if right then remove span, z-sort
  paint pixel
```

# Comparisons

**Z-Buffer and Scanline**
- + Exploit spatial coherence
  - Incremental sorting: bubble sort
  - Incremental computation: new x,y,z,c...

**Scanline**
- + Raster scan order; refresh processor
- + Deferred shading
- + Write-only algorithm
- + Easy to modify to do transparency

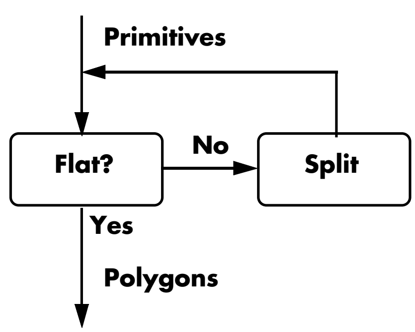
**Z-buffer**
- + Unlimited complexity
- + Diverse set of primitives



# Reyes Algorithm

**Adaptive subdivision of higher level primitives**

**Lane-Carpenter**



## Bucket Version

**Conserve memory!**

```
sort(primitive)
  bound = primitive->bound();
  if( inside(bound,view) )
    bucket[bound->min] = primitive;

render()
  for each bucket
    for each primitive in bucket
      if( primitive->flat() )
         draw(primitive->polygon());
      else
         sort(primitive->split());
```



## Grid Version

**Efficient shading of small primitives**