clipping
• last time: clipping a line segment
• can also clip triangles
  – restrict Δ bounding box to rendering window
  – easy solution
  – not required in p1
• speed up: demo

visibility
• issue: triangle visibility
• solution in p1: “painter’s algorithm”
  – draw Δ’s back to front
  – p1 uses heuristic to sort Δ’s
    (can see artifacts: demo)
  – Graphics cards: use depth buffer

jot/mlib
• for math intensive stuff (e.g. line clipping), use jot/mlib
• online documentation (see proj1 page)
• read header files
  – mlib/point2.H
  – mlib/vec2.H
  – mlib/points.H
• see: proj1/jot-tips.C

blackboard
computing 2D barycentric coordinates
using jot mlib
If \( \mathbf{a}, \mathbf{b}, \mathbf{c} \) are triangle vertices, \( \mathbf{p} \) is the pixel you
are evaluating (all in PIXEL coordinates), then the barycentric coordinates are:

\[
\alpha = \frac{\det(\mathbf{b} - \mathbf{p}, \mathbf{c} - \mathbf{p})}{\det(\mathbf{b} - \mathbf{a}, \mathbf{c} - \mathbf{a})}; \\
\beta = \frac{\det(\mathbf{p} - \mathbf{a}, \mathbf{c} - \mathbf{a})}{\det(\mathbf{b} - \mathbf{a}, \mathbf{c} - \mathbf{a})}; \\
\gamma = \frac{\det(\mathbf{b} - \mathbf{a}, \mathbf{p} - \mathbf{a})}{\det(\mathbf{b} - \mathbf{a}, \mathbf{c} - \mathbf{a})};
\]

OpenGL introduction
• name:
  – 1980’s: SGI’s graphics library: GL
  – basis for OpenGL API
• client/server model
• manages resources (GPU)
  – framebuffer
  – processor
  – memory
• API similar to DirectX
not included

- integration with media (e.g. audio, video…)
- UI widgets, windows, input handling
- high-level objects
  (e.g. meshes, scene graph)
- advanced algorithms, e.g.:
  – collision detection
  – physics
  – global illumination

what does it do?

- render 2D and 3D primitives or images
- rasterize Δ’s
- lighting (local illumination)
- texture mapping
- visibility

can add missing stuff

- UI widgets, windows, input handling:
  – via GLUT (as in discussions)
    - standard library
    - now freeglut
  – via GLUI (as in jot)
    - written by ex-Ph.D. student Paul Rademacher
- Meshes, scene graphs:
  – OpenSceneGraph
  – G3D
  – jot

typical program

- create window
- loop:
  – clear frame buffer
  – set state
  – setup lights, camera…
  – draw primitives
  – swap buffers

primitive processing pipeline

1. vertex processing
   – transformations: 3D 2D
   – lighting
2. clipping, primitive assembly
3. fragment processing
   – rasterize primitives
   – interpolate colors, texture coordinates, etc.
4. fragment text
   – depth, alpha

newly programmable parts

1. vertex processing
   – transformations: 3D 2D
   – lighting
2. clipping, primitive assembly
3. fragment processing
   – rasterize primitives
   – interpolate colors, texture coordinates, etc.
4. fragment text
   – depth, alpha
evolution of OGL

• originally pipeline was “fixed”
  – could set values, enable/disable state, etc.
• additions to OGL thru “extensions”
• extensions not guaranteed part of API
  – e.g., just supported by 1 vendor, or ARB
  – may be incorporated into later OGL version

now: GLSL

```c
void main() {
  // compute the vertex normal and position in eye coordinates:
  N = gl_NormalMatrix * gl_Normal;
  P = gl_ModelViewMatrix * gl_Vertex;
  // output vertex position in clip coordinates
  gl_Position = ftransform();
}
```

evolution of programmable OGL

• originally extensions to bypass fixed-functionality pipeline
• low-level assembly code, e.g.:
  ```c
  ! ! V P 1.0
  D P 4 R 0.x, v[OPOS], c[0];
  D P 4 R 0.y, v[OPOS], c[1];
  D P 4 R 0.z, v[OPOS], c[2];
  DP 4 R.0.w, v[OPOS], c[3];
  DP 4 M 1.x, R 0, c[4];
  DP 4 M 1.y, R 0, c[5];
  ...
  ```

GLSL relatively new

• available as extensions in OpenGL 1.5
• part of OpenGL 2.0
  – for project 2: work in newer labs!

example code

```c
// initialize window here...

// clear frame buffer
glClearColor(0,0,0,0);
glClear(GL_COLOR_BUFFER_BIT);

// set up camera
glOrtho(0,1,0,1,-1,1);
```

every example, cont’d

```c
// set current color
glColor3f(1,1,0);

// draw points
glPointSize(2.0); // must be before glBegin()
glBegin(GL_POINTS);
glVertex3f(x0,y0,z0);
glVertex3f(x1,y1,z1);
--
glEnd();
```
primitives

GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_TRIANGLES
GL_TRIANGLE_STRIP
...

(see blackboard, or red book)

more example

// draw triangles
glBegin(GL_TRIANGLE_STRIP);
   glColor3fv(c0);
   glNormal3fv(n0);
   glVertex3fv(v0);
   glColor3fv(c1);
   glNormal3fv(n1);
   glVertex3fv(v1);
...
   glEnd();

reading

• OpenGL programming guide (“red book”)
  – chapters 1 and 2

next up

• OpenGL lighting
• 2D and 3D transforms
• GLSL programming