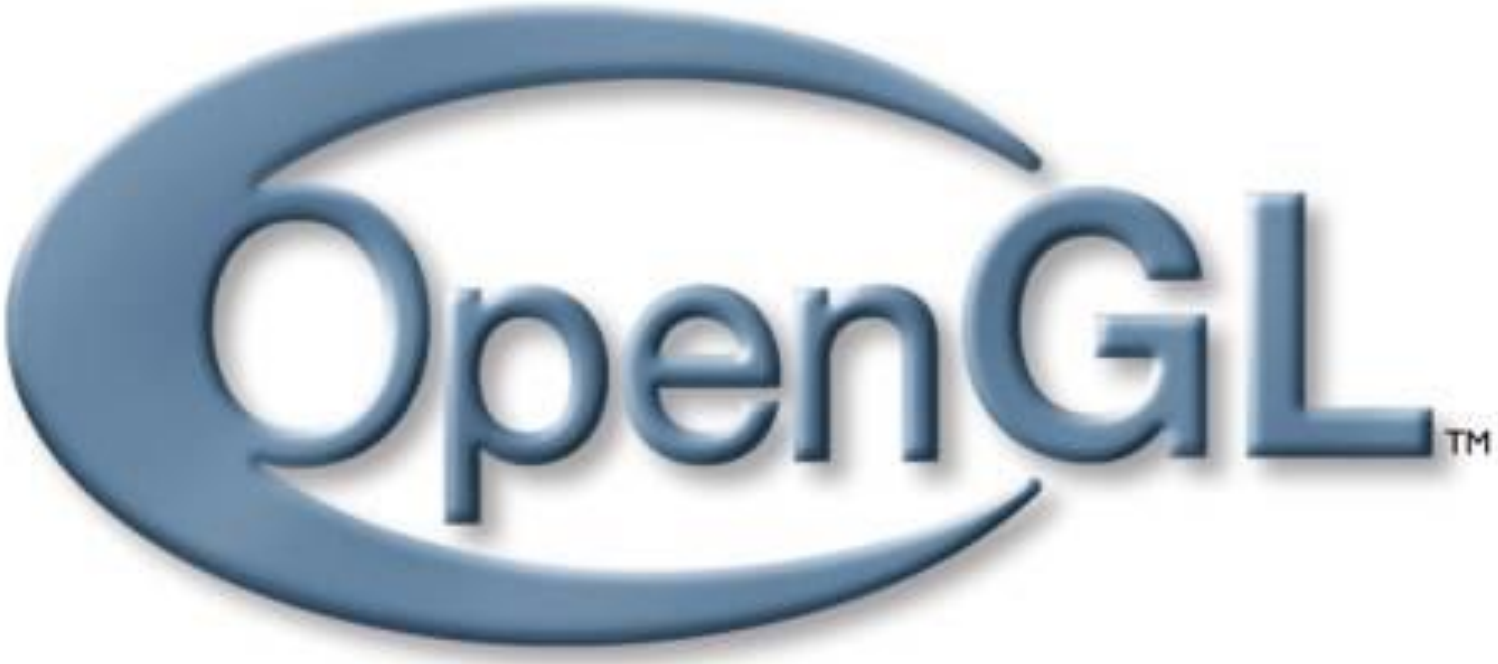


part 4

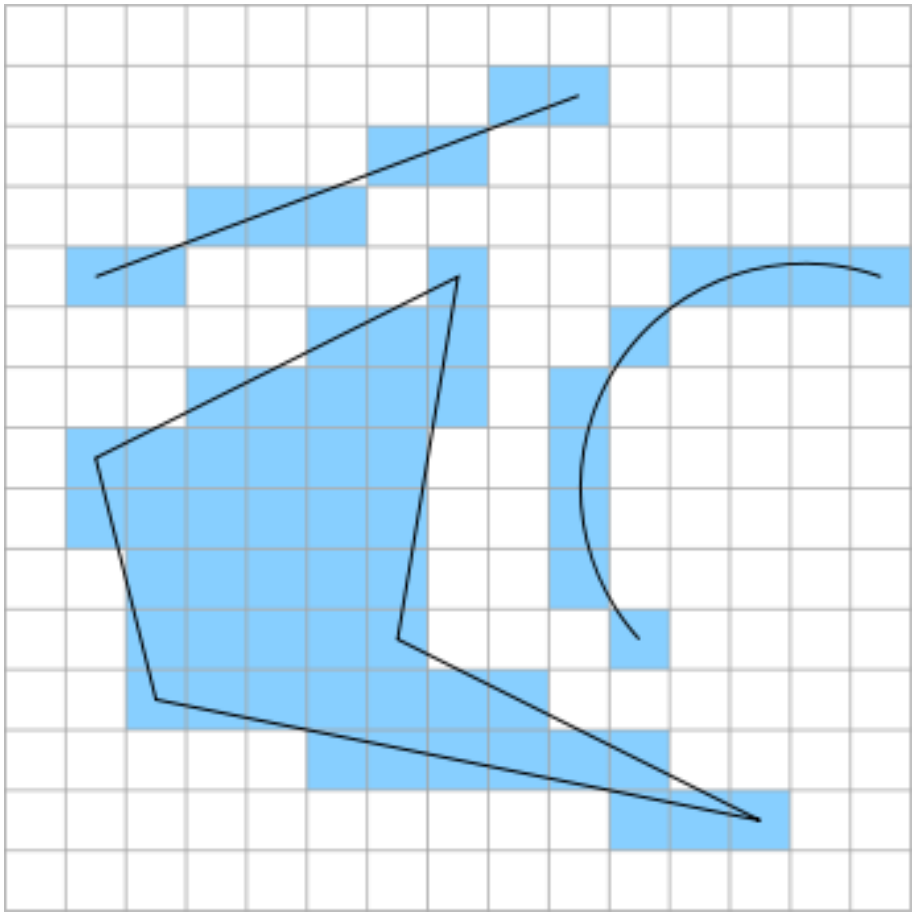
Martin Samuelčík
http://www.sccg.sk/~samuelcik
Room I4

After transformation

- Vertices have now window coordinates
- Vertices still form geometric primitives – points, segments, simple polygons
- We need to render primitives on the screen – set proper color of window pixels under primitives
- We need to break primitive into set of fragments, for each pixel one fragment

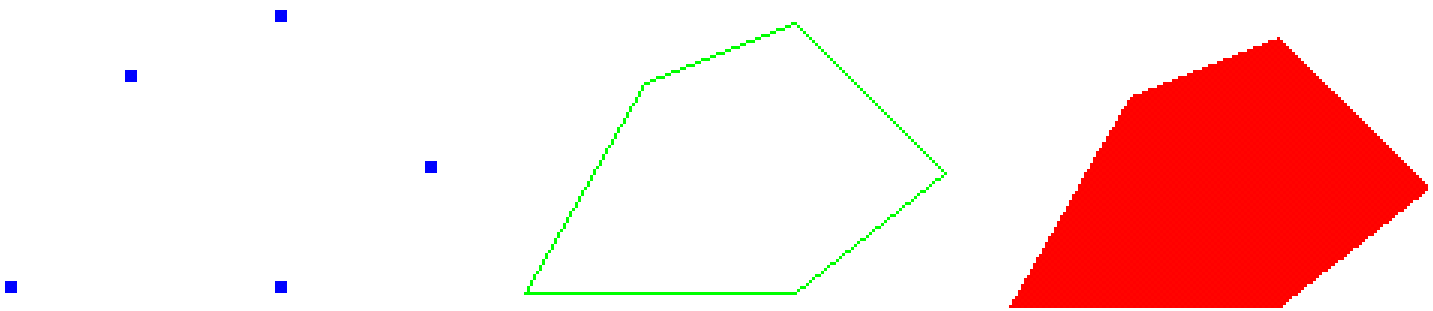
Rasterization

- Input: point or segment or polygon
- Output: set of fragments – to be pixels



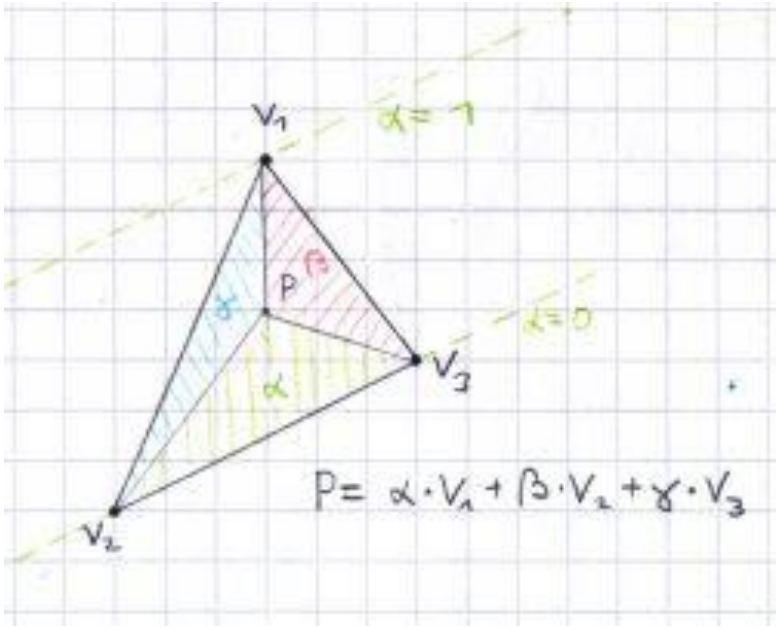
Rasterization

- Setting how polygons are rasterized
- **void glPolygonMode(GLenum face, GLenum mode)**
  - face - GL_FRONT_AND_BACK for front and back facing polygons
  - mode - GL_POINT, GL_LINE, and GL_FILL



Rasterization

- Each new fragment must have computed same attributes as vertices defining primitive
- Computing barycentric coordinates of fragment inside primitive
- Using barycentric coordinates for interpolation of attribute values (color, texture coordinates, depth)

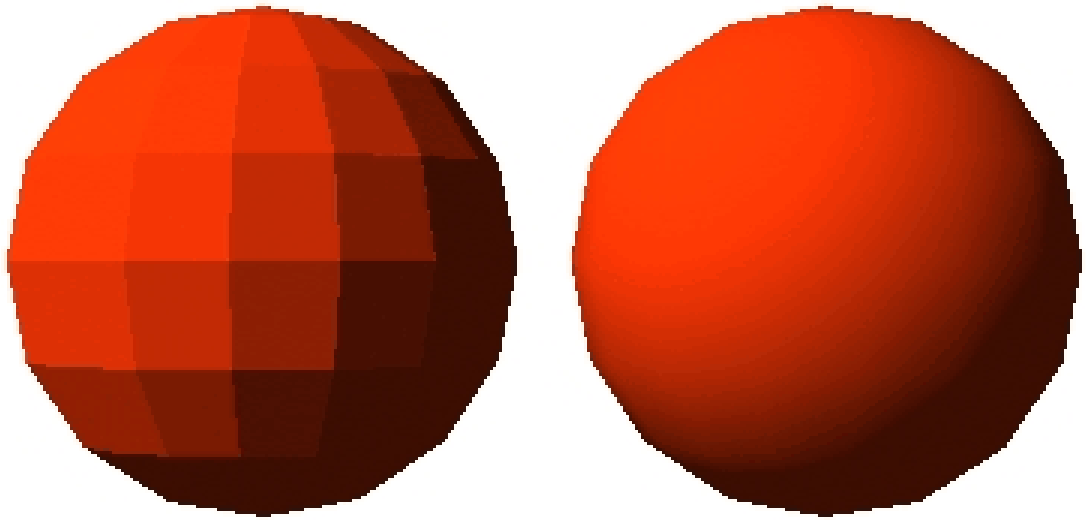


$$P = V_1 \cdot \alpha + V_2 \cdot \beta + V_3 \cdot \gamma$$
$$\alpha + \beta + \gamma = 1$$

$$\alpha(P) = \frac{O_1(P)}{O_1(V_1)}$$
$$\beta(P) = \frac{O_2(P)}{O_2(V_2)}$$
$$\gamma(P) = \frac{O_3(P)}{O_3(V_3)}$$

Color interpolation

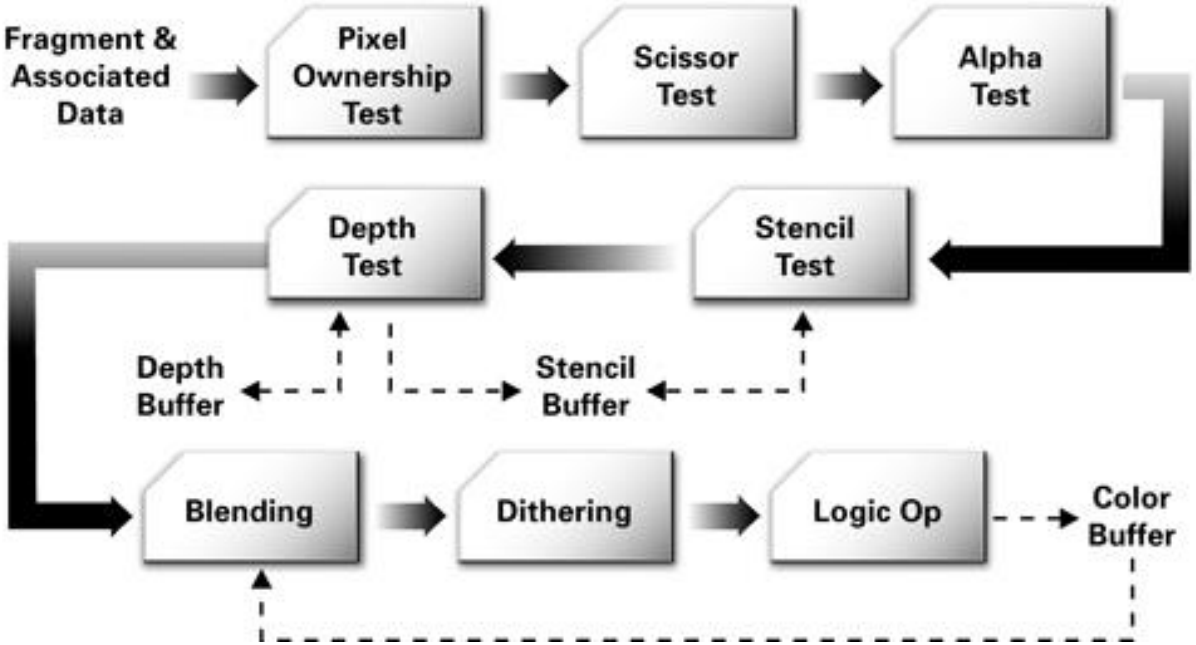
- Primitive has colors as attributes in vertices
- For each fragment, interpolate vertices colors using barycentric coordinates
- **glShadeModel(GLenum mode)**
  - mode - GL_FLAT, GL_SMOOTH



Flat Gouraud

Fragment processing

- Rasterizer generates set of fragments
- On fragments, several operations and tests are performed
- If fragment passes all test, its attributes are written into appropriate buffers

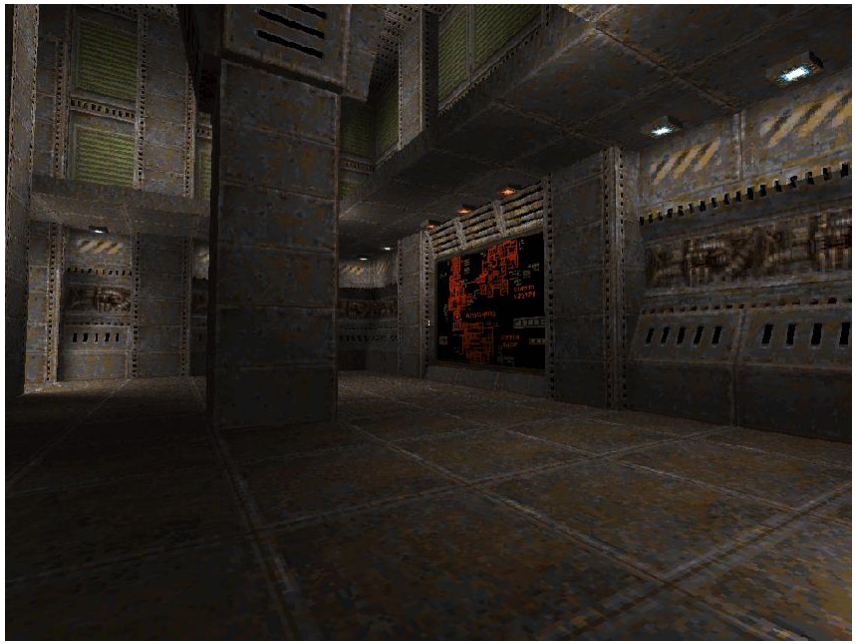


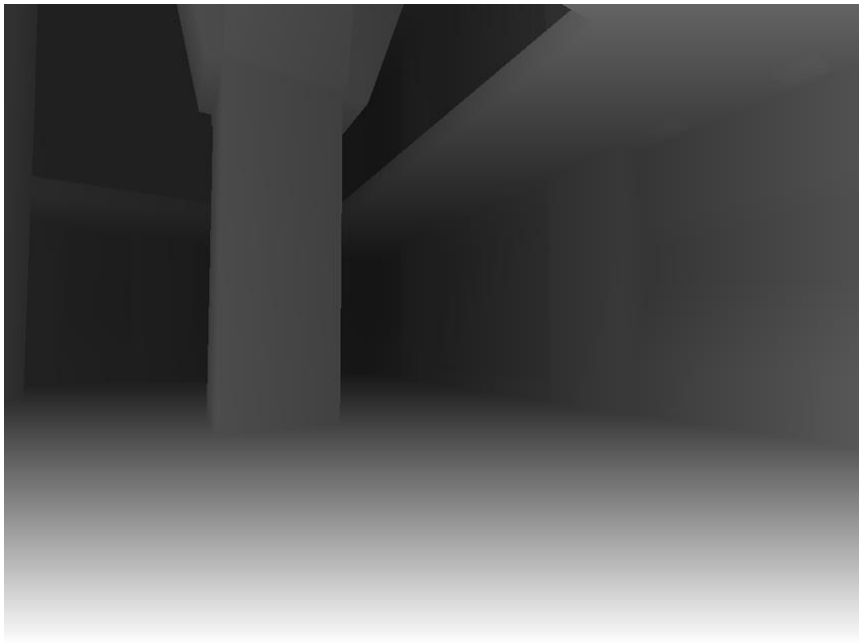
Buffers

- Holds informations about pixels
- Color buffers
  - Content of color buffer is transferred to the render window and displayed
  - Double buffering: Front and back buffer
  - Double buffering with stereo: front-left, front-right, back-left, back-right,
  - Any number of auxiliary color buffers
- Depth buffer
  - Depth values for each visible pixel, used for visibility determination
- Stencil Buffer
  - Byte values used for masking
- Accumulation buffer
  - Buffer used for accumulating several frames

Framebuffer

- Term for all buffers together
- Each buffer has same pixel dimensions, same width and height in pixels





Double buffering

- One color buffer for rendering (writing color from fragment to buffer)
- One color buffer for displaying on screen
- **glutSwapBuffers()** – swapping two color buffers, rendered content is immediately displayed on screen
- Setting color buffer for reading and writing
- **Void glReadBuffer(GLenum mode)**
  - mode - GL_FRONT, GL_BACK, default is GL_BACK
- **void glDrawBuffer(GLenum mode)**
  - mode - GL_FRONT, GL_BACK, default is GL_BACK

Setting Buffers

- Buffer properties are tied with operating system, hardware and drivers
- Set when OpenGL window is created
- void glutInitDisplayMode(unsigned int mode)
  - GLUT_RGBA
  - GLUT_RGB
  - GLUT_SINGLE
  - GLUT_DOUBLE
  - GLUT_ACCUM
  - GLUT_ALPHA
  - GLUT_DEPTH
  - GLUT_STENCIL
  - GLUT_MULTISAMPLE
  - GLUT_STEREO

Advanced setting

- Setting number of bits for each buffer
- Using one string containing switches
- **glutInitDisplayString(const char* displayMode)**
- Example: glutInitDisplayString ("stencil>=2 rgb8 double depth>=16")
  - Using 2 color buffers, front and back
  - Each color buffer has 24 color depth per pixel
  - Each pixel of depth buffer has precision at least 16 bits
  - Each pixel of stencil buffer has precision at least 2 bits

Buffers precision

- Retrieving pixel precision of each buffer
- **void glGetIntegerv(GLenum pname, GLint * params)**

Parameter	Meaning
GL_RED_BITS, GL_GREEN_BITS, GL_BLUE_BITS, GL_ALPHA_BITS	Number of bits per R, G, B, or A component in the color buffers
GL_DEPTH_BITS	Number of bits per pixel in the depth buffer
GL_STENCIL_BITS	Number of bits per pixel in the stencil buffer
GL_ACCUM_RED_BITS, GL_ACCUM_GREEN_BITS, GL_ACCUM_BLUE_BITS, GL_ACCUM_ALPHA_BITS	Number of bits per R, G, B, or A component in the accumulation buffer

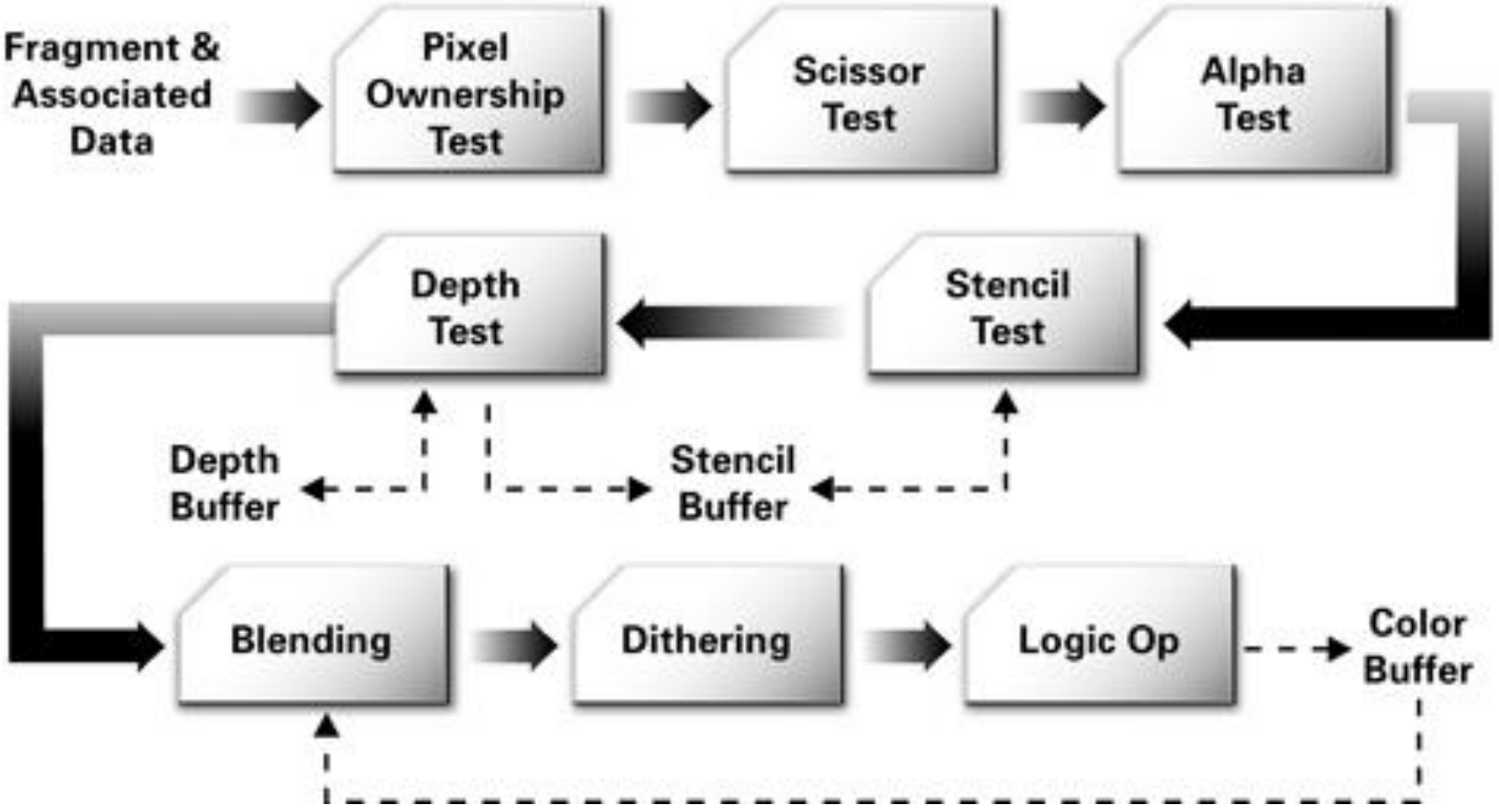
Clearing buffers

- Clearing = filling whole buffer with one value
- Clearing value is defined for each type of buffer as state variable
  - **void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)**
  - **void glClearDepth(GLclampd depth)**
  - **void glClearStencil(GLint s)**
  - **void glClearAccum(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)**
- Actual clearing: **glClear(GLbitfield mask)**
  - mask = GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_STENCIL_BUFFER_BIT, GL_ACCUM_BUFFER_BIT

Masking buffers

- Sets the masks used to control writing into the buffers – enable or disable writing
- **void glColorMask(GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha)**
  - Enable or disable writing to each color channel of color buffer
- **void glDepthMask(GLboolean flag)**
  - Enable or disable writing to depth buffer
- **void glStencilMask(GLuint mask)**
  - Use bit bask when writing to stencil buffer

Fragment operations



Scissor test

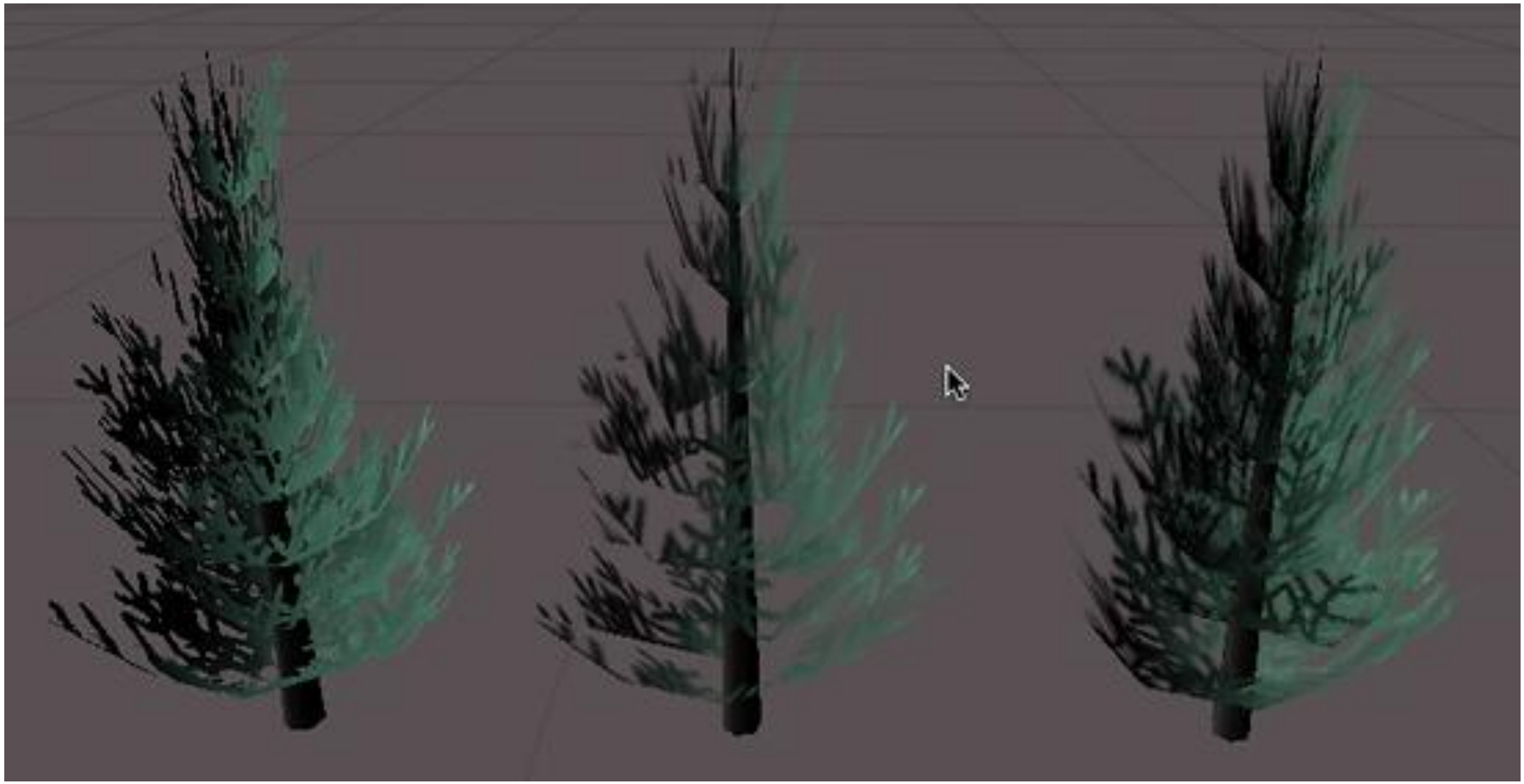
- Discard fragment with window coordinates out of given rectangle
- Restrict drawing to rectangular portion of window, affects also glClear
- Setting rectangle - **void glScissor(GLint x, GLint y, GLsizei width, GLsizei height)**
- Enable or disable with
  glEnable(GL_SCISSOR_TEST)
  glDisable(GL_SCISSOR_TEST)

Alpha test

- Discard fragment when alpha value in its color does not match selected criteria
- Restrict drawing of pixel if condition fails
- Used for basic transparency with pixels opaque or transparent
- **void glAlphaFunc(GLenum func, GLclampf ref);**
  - func = GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER, GL_NOTEQUAL
  - ref in <0,1>
- Enable or disable

  glEnable(GL_ALPHA_TEST)
  glDisable(GL_ALPHA_TEST)

Alpha test



Stencil test

- For each fragment compare reference value with value from fragment position in stencil buffer
- Based on result of comparison, several operations are allowed
  - Discard fragment
  - Modify appropriate value in stencil buffer
- Used for masking part of screen, only in this part we will render
- Usually multiple passes are needed

Stencil test

- Comparing value from stencil buffer and reference value
- Enable or disable

  glEnable(GL_STENCIL_TEST)
  glDisable(GL_STENCIL_TEST)

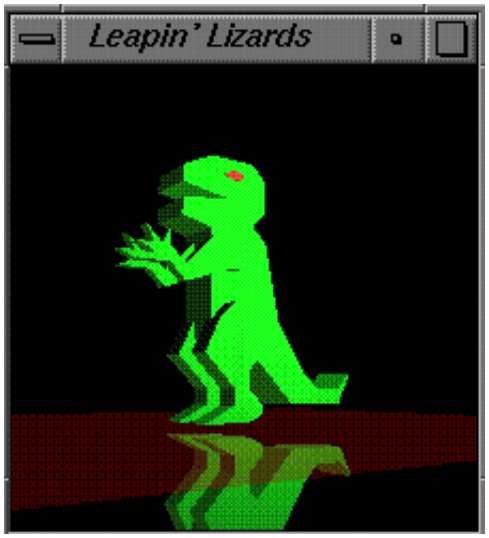
- **void glStencilFunc(GLenum func, GLint ref, GLuint mask)**
  - func = GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER, or GL_NOTEQUAL
  - ref = reference value for testing
  - mask = mask for bitwise AND before testing

Stencil operation

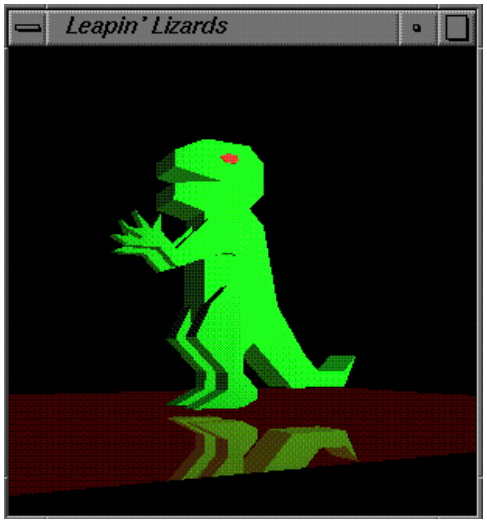
- What to do with appropriate stencil value based on result of stencil and depth (consecutive) test
- **void glStencilOp(GLenum fail, GLenum zfail, GLenum zpass)**
  - fail – do something when stencil test fails
  - zfail – do something when stencil test passes, depth test fails
  - zpass – do something when stencil test passes, depth test passes
  - fail, zfail, zpass = GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_DECR, GL_INVERT
  - keeping current value, replacing with zero, replacing with reference value, incrementing, decrementing, and bitwise-inverting

Example

```
/* Disable writing to color and depth buffers */
/* We will write into stencil buffer only */
glDisable(GL_DEPTH_TEST);
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);

/* Draw floor into the stencil buffer. */
glEnable(GL_STENCIL_TEST);
glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);
glStencilFunc(GL_ALWAYS, 1, 0xffffffff);
drawFloor();

/* Re-enable update of color and depth. */
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glEnable(GL_DEPTH_TEST);

/* Now, only render where stencil is set to 1. */
glStencilFunc(GL_EQUAL, 1, 0xffffffff);  /* draw if ==1 */
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

/* Draw the reflected object only in floor pixels. */
glScalef(1.0f, 1.0f, -1.0f);
drawDino();
glDisable(GL_STENCIL_TEST);

/* Draw transparent floor */
glEnable(GL_BLEND);
glColor4f(1.0f, 0.1f, 0.1f, 0.5f);
drawFloor();
glDisable(GL_BLEND);

/* Draw object normally*/
drawDino();
```



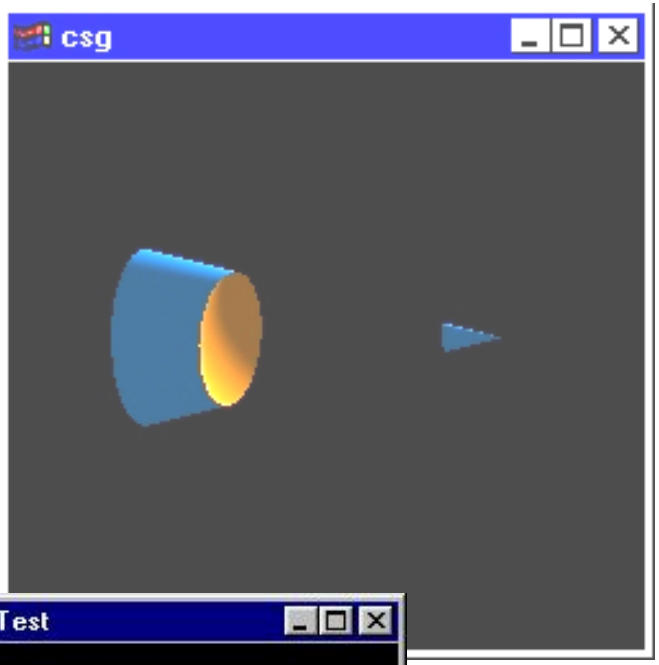
no stencil buffer

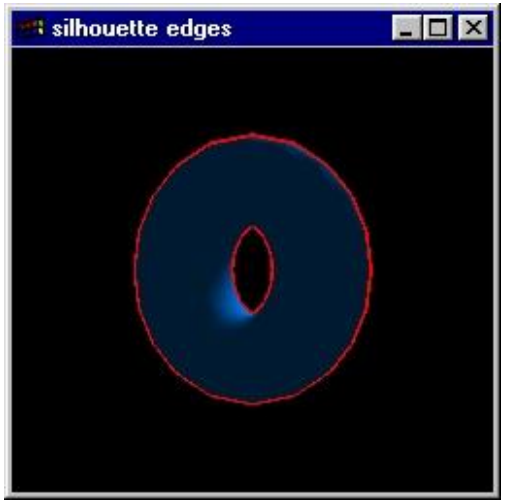


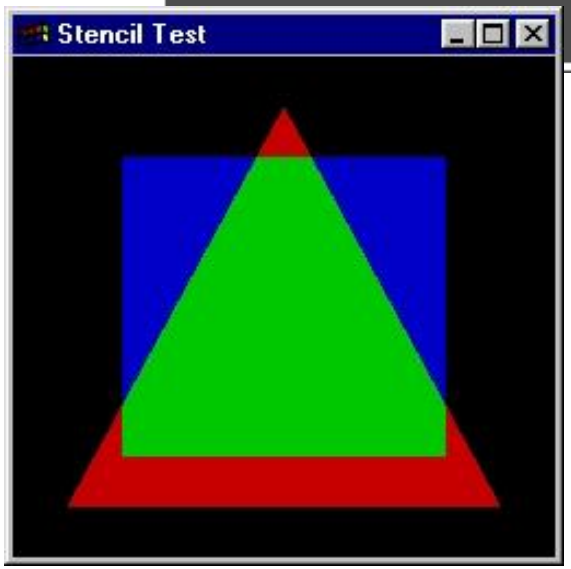
with stencil buffer

OpenGL

Examples







OpenGL®

Depth test

- Discard fragment id comparison of fragment's depth and appropriate value in depth buffer fails
- Depth of fragment is computed by transformations and by interpolation in rasterizer
- Setting comparison **void glDepthFunc(GLenum func);**
  - func = GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER, GL_NOTEQUAL
- Enable or disable

  glEnable(GL_DEPTH_TEST)
  glDisable(GL_DEPTH_TEST)

Blending

- When writing fragment's color to color buffer – instead do simple mix of fragment's color and appropriate buffer's color
- Creating translucent fragments, because we see old color as well as new color
- Used for transparency, digital composition, painting
- Using alpha values

Blending

- Fragment color – source - (R_s,G_s,B_s,A_s)
- Color in buffer – destination - (R_d,G_d,B_d,A_d)
- Final color written into color buffer – $(R_sS_r+R_dD_r,G_sS_g+G_dD_g,B_sS_b+B_dD_b,A_sS_a+A_dD_a)$
- Setting blending factors
- void **glBlendFunc(GLenum sfactor, GLenum dfactor)**
  - sfactor – setting S_r,S_g,S_b,S_a
  - dfactor – setting D_r,D_g,D_b,D_a
  - Default: GL_ONE, GL_ZERO
- Enable or disable (disabled by default)

  glEnable(GL_BLEND)

  glDisable(GL_BLEND)

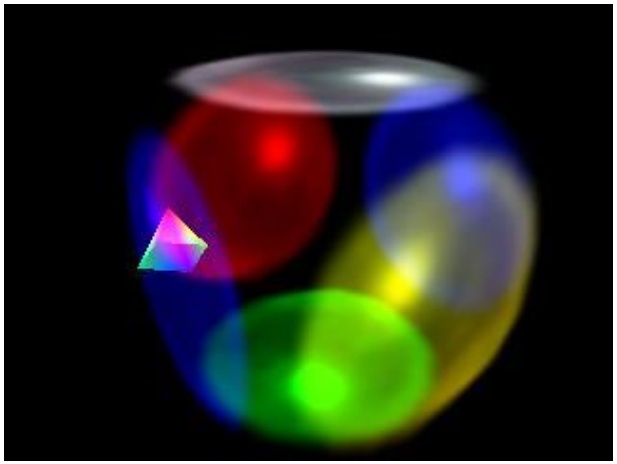
Blending factors

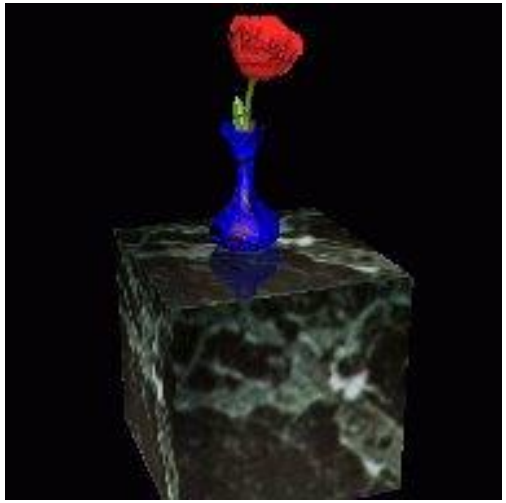
Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	source or destination	(0, 0, 0, 0)
GL_ONE	source or destination	(1, 1, 1, 1)
GL_DST_COLOR	source	(R_d, G_d, B_d, A_d)
GL_SRC_COLOR	destination	(R_s, G_s, B_s, A_s)
GL_ONE_MINUS_DST_COLOR	source	$(1, 1, 1, 1)-(R_d, G_d, B_d, A_d)$
GL_ONE_MINUS_SRC_COLOR	destination	$(1, 1, 1, 1)-(R_s, G_s, B_s, A_s)$
GL_SRC_ALPHA	source or destination	(A_s, A_s, A_s, A_s)
GL_ONE_MINUS_SRC_ALPHA	source or destination	$(1, 1, 1, 1)-(A_s, A_s, A_s, A_s)$
GL_DST_ALPHA	source or destination	(A_d, A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPHA	source or destination	$(1, 1, 1, 1)-(A_d, A_d, A_d, A_d)$
GL_SRC_ALPHA_SATURATE	source	$(f, f, f, 1)$; $f=\min(A_s, 1-A_d)$

Blending in 3D

- When rendering transparent object, all objects behind it must be already rendered
- Transparent objects usually drawn back to front
- Render opaque objects first with enabled depth buffer
- Make the depth buffer read-only while drawing the translucent objects
- Additive blending – no ordering needed, but energy is not preserved
  - glBlendFunc (GL_SRC_ALPHA, GL_ONE)

Examples





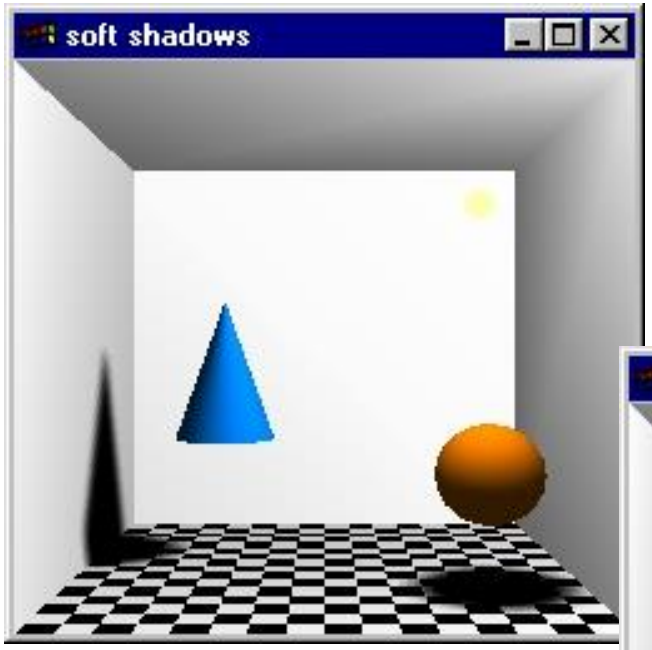
Accumulation buffer

- Used for accumulating values from color buffers
- **void glAccum(GLenum op, GLfloat value)**
  - op = GL_ACCUM reads each pixel from the buffer current read buffer, multiplies the R, G, B, and alpha by value, and adds the result to the accumulation buffer.
  - op = GL_LOAD does the same thing, except that the values replace those in the accumulation buffer rather than being added to them.

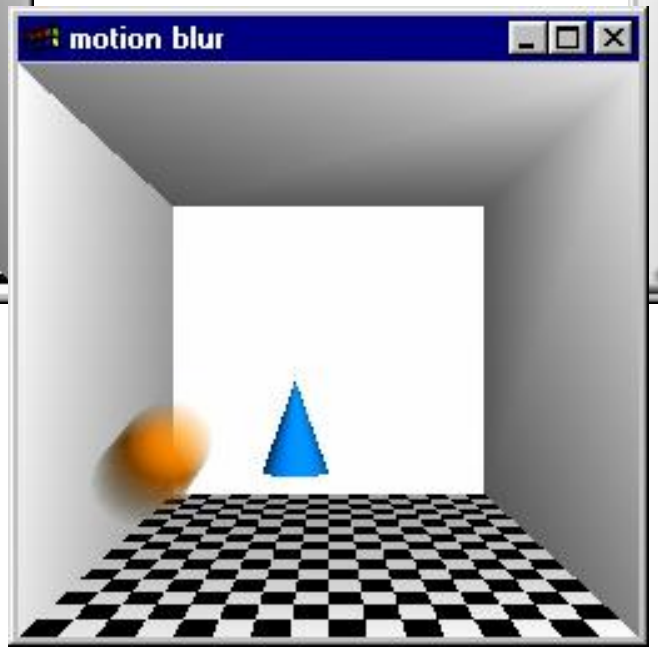
Accumulation buffer

- **void glAccum(GLenum op, GLfloat value)**
  - op = GL_RETURN takes values from the accumulation buffer, multiplies them by value, and places the result in write buffer(s).
  - op = GL_ADD and GL_MULT simply add or multiply the value of each pixel in the accumulation buffer by value and then return it to the accumulation buffer. For GL_MULT, value is clamped to be in the range [-1.0,1.0]. For GL_ADD, no clamping occurs.

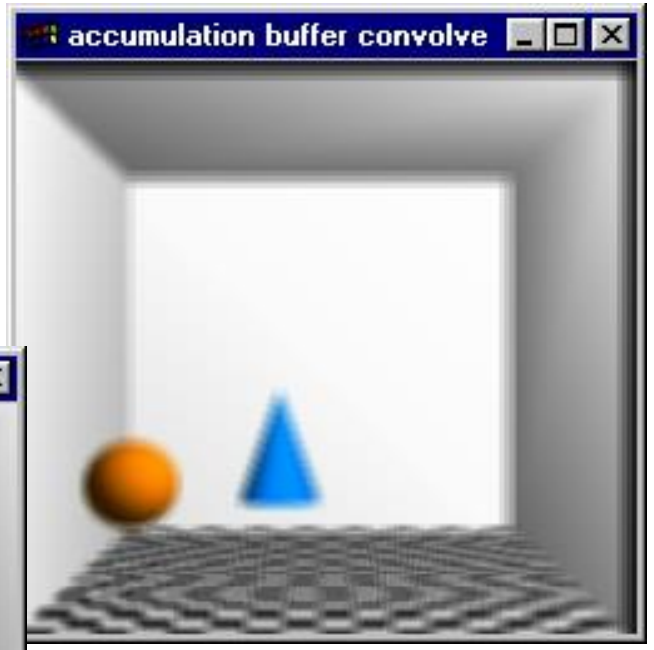
Examples



Soft shadows



Motion blur



Convolution, antialiasing

The End!

Questions?