Blending: Intro

- Simulates translucency/opacity
- Occurs after rasterization and fragmentation
  - Last thing before writing pixels to buffer
- Combines color of incoming fragment (source) with value already in buffer (destination)
- Uses alpha value (4th color argument), which represents opacity
  - Higher values represent greater opacity (1.0 $\Rightarrow$ 100% opaque)

glEnable(GL_BLEND);
void glBlendFunc (GLenum sfactor, Glenum dfactor);
void glBlendFuncSeparate (GLenum sRGB, Glenum dRGB, GLenum sAlpha, Glenum dAlpha);

Same as $glBlendFunc$ but allows separate control of alpha blending

- Computing pixel color
  - Let (S_r, S_g, S_b, S_a) represent blending function for source
  - Let (D_r, D_g, D_b, D_a) represent blending function for dest
  - Pixel color = $(R_s * S_r + R_d * D_r,\ G_s * S_g + G_d * D_g,\ B_s * S_b + B_d * D_b,\ A_s * S_a + A_d * D_a)$
  - Result clamped to $[0, 1]$

Blending: Intro (2)

- Built-in blending functions

Constant	RGB Blend Factor	Alpha Blend Factor
GL_ZERO	(0,0,0,0)	0
GL_ONE	(1,1,1,1)	1
GL_DST_COLOR	(R_d, G_d, B_d, A_d)	A_d
GL_SRC_COLOR	(R_s, G_s, B_s, A_s)	A_s
GL_ONE_MINUS_DST_COLOR	(1,1,1,1) - (R_d, G_d, B_d, A_d)	$1 - A_d$
GL_ONE_MINUS_SRC_COLOR	(1,1,1,1) - (R_s, G_s, B_s, A_s)	$1 - A_s$
GL_SRC_ALPHA	(A_s, A_s, A_s, A_s)	A_s
GL_ONE_MINUS_SRC_ALPHA	(1,1,1,1) - (A_s, A_s, A_s, A_s)	$1 - A_s$
GL_DST_ALPHA	(A_d, A_d, A_d, A_d)	A_d
GL_ONE_MINUS_DST_ALPHA	(1,1,1,1) - (A_d, A_d, A_d, A_d)	$1 - A_d$
GL_SRC_ALPHA_SATURATE	(f,f,f,1); f = $\min(A_s, 1 - A_d)$	1
GL_CONSTANT_COLOR	(R_c, G_c, B_c, A_c)	A_c
GL_ONE_MINUS_CONSTANT_COLOR	(1,1,1,1) - (R_c, G_c, B_c, A_c)	$1 - A_c$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c, A_c)	A_c
GL_ONE_MINUS_CONSTANT_ALPHA	(1,1,1,1) - (A_c, A_c, A_c, A_c)	$1 - A_c$

where
d represents destination component,
s represents source component, and
c represents a constant component

Blending: Intro (3)

void glBlendColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a);

Sets values of *constant color* for blending

void glBlendEquation (GLenum mode);

Allows operations other than addition for combining source and destination components

Mode	Computed Factor
GL_FUNC_ADD	$C_sS + C_dD$
GL_FUNC_SUBTRACT	$C_sS - C_dD$
GL_FUNC_REVERSE_SUBTRACT	$C_dD - C_sS$
GL_MIN	$min(C_sS, C_dD)$
GL_MAX	$max(C_sS, C_dD)$
GL_LOGIC_OP	S op D

where C_s, C_d represent source and destination colors, respectively

void glLogicOp(GLenum opcode);

Combines source and destination components using logical operations

OpCode	Operation
GL_CLEAR	0
GL_COPY	s
GL_NOOP	d
GL_SET	1
GL_COPY_INVERTED	$\neg s$
GL_INVERT	$\neg d$
GL_AND_REVERSE	$s \wedge \neg d$
GL_OR_REVERSE	$s \vee \neg d$
GL_AND	$s \wedge d$
GL_OR	$s \vee d$
GL_NAND	$\neg(s \wedge d)$
GL_NOR	$\neg(s \vee d)$
GL_XOR	$s \otimes d$
GL_EQUIV	$\neg(s \otimes d)$
GL_AND_INVERTED	$\neg s \wedge d$
GL_OR_INVERTED	$\neg s \vee d$

Blending: Intro (4)

- Common combinations of alpha values for blending

1. Default

Source	Dest
GL_ONE	GL_ZERO

2. Two equally blended images

Image	Source	Dest	Alpha
first	GL_ONE	GL_ZERO	
second	GL_SRC_ALPHA	GL_ONE_MINUS_SRC_ALPHA	0.5

3. Three equally blended images

Image	Source	Dest	Alpha
all	GL_SRC_ALPHA	GL_ONE	0.333...

4. Multiple overlapping translucent images
   - Draw from farthest to nearest with specified alpha

Image	Source	Dest	Alpha
background	GL_ONE	GL_ZERO	
first	GL_SRC_ALPHA	GL_ONE_MINUS_SRC_ALPHA	as appropriate
second	GL_SRC_ALPHA	GL_ONE_MINUS_SRC_ALPHA	as appropriate
...	...	...	...

5. Filtering specific colors
   - source = GL_DST_COLOR or GL_ONE_MINUS_DST_COLOR,
   - dest = GL_SRC_COLOR or GL_ONE_MINUS_SRC_COLOR
6. Billboarding - pasting one image on top of another
   - Use alpha = 0 for fragments you don't want to see; alpha = 1 for those you do

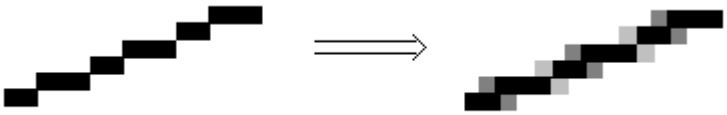
Blending: Intro (5)

- Blending and depth
  - When use depth buffering, will affect whether fragment is displayed or not
  - Will inhibit blending of translucent objects that are deeper than current depth
  - To deal with the problem:
    1. Enable depth buffering
    2. Draw opaque objects as usual
    3. Make depth buffer read-only

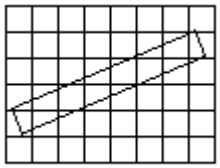
       void glDepthMask(GLboolean value);
       GL_FALSE makes read-only
    4. Draw translucent objects

Blending: Antialiasing

- Staircasing effect of non-vertical and non-horizontal lines called **aliasing**
- **Antialiasing**: decreasing staircasing effect



- Use blending to achieve:
  - Blend fragment with each pixel it overlaps
  - Use per cent of overlap to determine amount of blending
  - Multiply alpha value by per cent coverage



- Antialiasing must be enabled

GL_POINT_SMOOTH
GL_LINE_SMOOTH
GL_POLYGON_SMOOTH

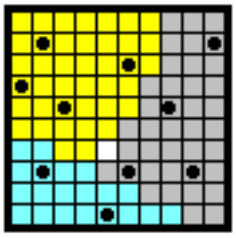
- Blending must be enabled
  - Alpha values for antialiasing points and lines:

    source = GL_SRC_ALPHA
    dest = GL_ONE_MINUS_SRC_ALPHA
  - For polygons:

    source = GL_SRC_ALPHA_SATURATE
    dest = GL_ONE
    - Draw polys front to back

Blending: Antialiasing (2)

- Multisampling
  - In multisampling, a given fragment now consists of multiple colors, depths, etc. dependent on the number of samples
    - These components result by dividing a pixel into subpixels and examining the values at random subpixels
    - These values will be different if objects partially overlay a pixel



  - Samples stored in a multisample buffer
  - Better for polygon antialiasing as do not have to sort polygons
  - Steps in using:
    1. Initialize glut for multisampling (sets up multisampling buffer)

       glutInitDisplayMode(...|GLUT_MULTISAMPLE|...);

    2. Verify that multisampling supported

       glGetIntegerv(GL_SAMPLE_BUFFERS, &bufs);
       glGetIntegerv(GL_SAMPLES, &samples);

       If $buffers == 1$, and $samples > 1$, can use multisampling
    3. Enable multisampling

       glEnable(GL_MULTISAMPLE);

Blending: Antialiasing (3)

– Standard multisampling does not incorporate alpha values

  * To incorporate alpha values enable one of

    GL_SAMPLE_ALPHA_TO_COVERAGE (uses fragment alpha value)
    GL_SAMPLE_ALPHA_TO_ONE (sets fragment alpha value to one and uses that value)
    GL_SAMPLE_COVERAGE (ANDs value set by following function)

    void glSampleCoverage(GLclampf value, GLboolean invert);
      value used to interpret alpha values when above modes enabled (except for TO_ONE mode)
      invert performs bitwise inversion of *value* before combining

– Multisampling precludes GL_SMOOTH operations

Blending: Fog

- Used to describe atmospheric effects
- Objects fade into distance
- Used to
  1. Create atmospheric effects
  2. Emphasize objects in foreground
  3. Increase rendering efficiency
- Must be enabled: *glEnable(GL_FOG);*
- Final color defined by $C = fC_i + (1 - f)C_f$, where

  C_i is color of incoming fragment, C_f is color of fog, and f represents the fog function
- The fog function f is set using
  glFogif(GLenum pname, TYPE param);
  glFogifv(GLenum pname, TYPE *params);

Pname	param
GL_FOG_MODE	GL_EXP $f = e^{-(density*z)}$
	GL_EXP2 $f = e^{-(density*z)^2}$
	GL_LINEAR $f = \frac{end-z}{end-start}$
GL_FOG_DENSITY	$0 \leq density \leq 1$
GL_FOG_START	$0 \leq start \leq 1$
GL_FOG_END	$0 \leq end \leq 1$
GL_FOG_COLOR	(R, G, B, A)
GL_FOG_INDEX	table index
GL_FOG_COORD_SRC	GL_FOG_COORD

where z is distance from eye to vertex

void glFogCoord*(TYPE z);
void glFogCoord*v(TYPE *z);

Allow explicit setting of z value on a per-vertex basis
Used in conjunction with GL_FOG_COORD_SRC

Blending: Point Parameters

- May want small points that can change in size and brightness
- May not want to model as 3D object
- Could model using glPointSize, but not valid between glBegin/glEnd pairs
- Functionality provided by

```
void glPointParameterf(GLenum pname, GLfloat param);
void glPointParameterfv(GLenum pname, GLfloat *param);
```

Pname	param
GL_POINT_DISTANCE_ATTENUATION	(a, b, c)
GL_POINT_SIZE_MIN	float
GL_POINT_SIZE_MAX	float
GL_FADE_THRESHOLD_SIZE	$0 \leq d \leq 1$
GL_POINT_SPRITE_COORD_ORIGIN	GL_LOWER_LEFT, GL_UPPER_LEFT

- Attenuation calculated as

$$derivedSize = clamp\left(size * \sqrt{\frac{1}{a + bd + cd^2}}\right)$$

  where d is distance from eye to point

- *fade threshold* specifies a different size threshold when multisampling enabled

$$fade = \left(\frac{derivedSize}{threshold}\right)^2$$

- Sprite coordinates specify origin location and direction of variation of texture coordinate t

Blending: Polygon Offset

- **Stitching** is undesirable effect produced when a polygon outline is overlaid onto a filled polygon
  - The outline may irregularly move from in front to behind the filled part
  - Caused by fact that depth calculated differently for lines and polygons
- To resolve, offset the polygon from the outline
  - Enable one of

GL_POLYGON_OFFSET_FILL
GL_POLYGON_OFFSET_LINE
GL_POLYGON_OFFSET_POINT

    for respective polygon drawing mode
- Offset specified using

  glPolygonOffset(GLfloat factor, GLfloat units);

  $offset = m * factor + r * units$, where

  m = maximum depth slope of polygon
  r is implementation-specific constant

  - Offset amount added to depth value of fragment

OpenGL Blending: Hints

- **Hint** is a suggestion to OpenGL to perform a bit of processing in a particular way
  - Only a suggestion to the implementation
  - May be handled differently in different implementations
  - Not guaranteed to have any effect

  void glHint (GLenum target, GLenum hint);

Target
GL_POINT_SMOOTH_HINT
GL_LINE_SMOOTH_HINT
GL_POLYGON_SMOOTH_HINT
GL_FOG_HINT
GL_PERSPECTIVE_CORRECTION_HINT

  - SMOOTH targets control amount of sampling during antialiasing
  - FOG determines whether calculations performed per pixel or per fragment
  - PERSPECTIVE determines how colors and textures are interpolated:
    1. Linearly, in screen space, or
    2. Using perspective projection in 3D

Hint
GL_NICEST
GL_FASTEST
GL_DONT_CARE