Multi-Pass Shading

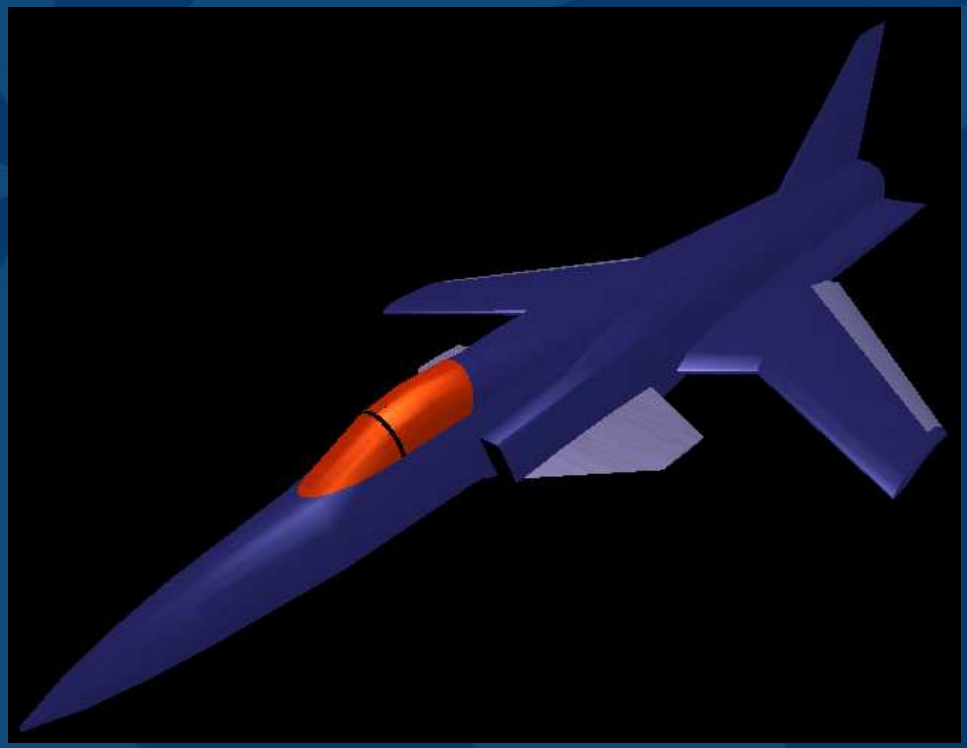
Marc Olano
SGI

Interactive Rendering

SAN ANTONIO SIGGRAPH 2002

Illusion of presence

- 10 – 30 – 60 frames per second
- Immediate response
- Simple appearance

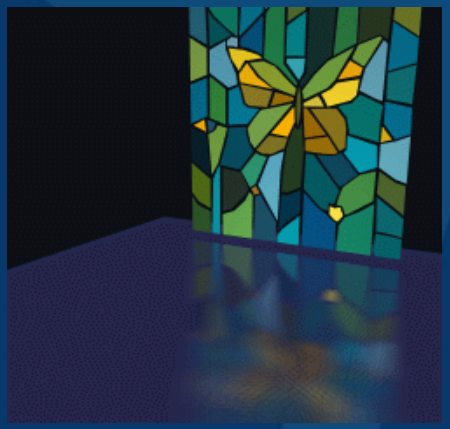


Multi-pass Rendering

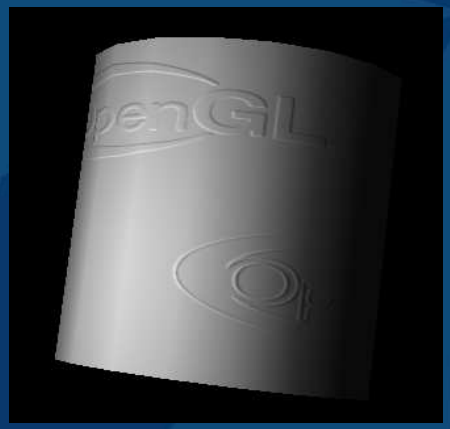
SAN ANTONIO SIGGRAPH 2002

Improved appearance

- Build effects
- Per-frame or per-object
- Still interactive



[Diefenbach97]



[Peercy97]

[Cabral99]

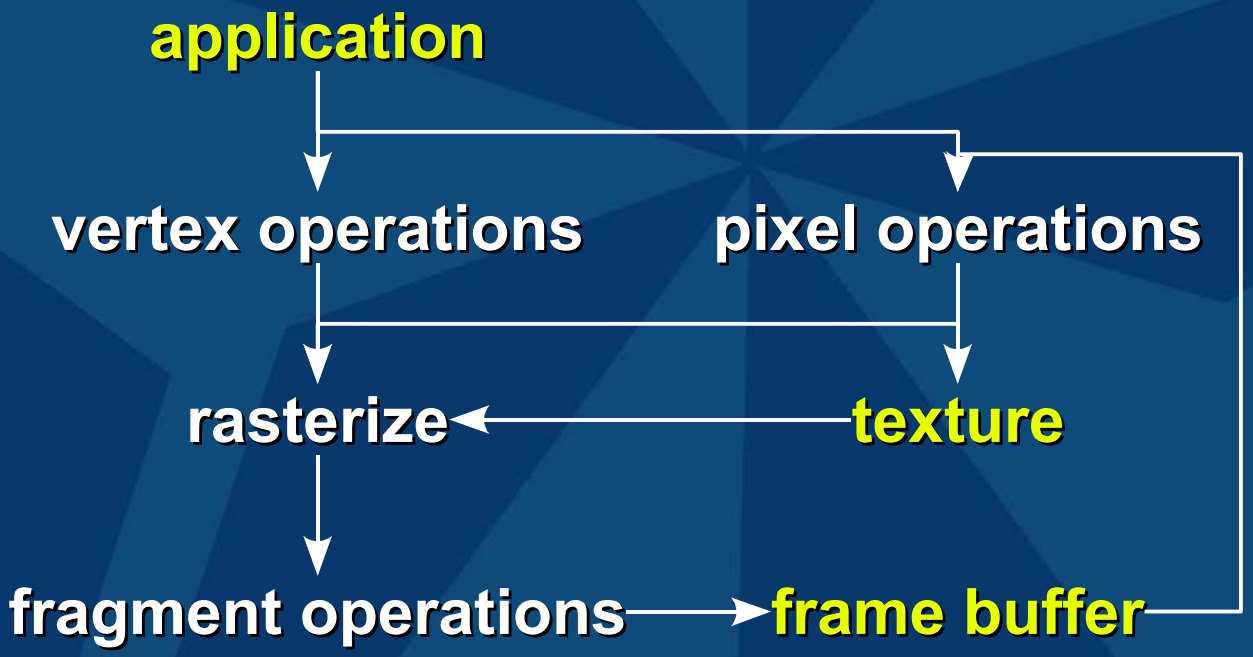
[Kautz99]

What's in a Pass?

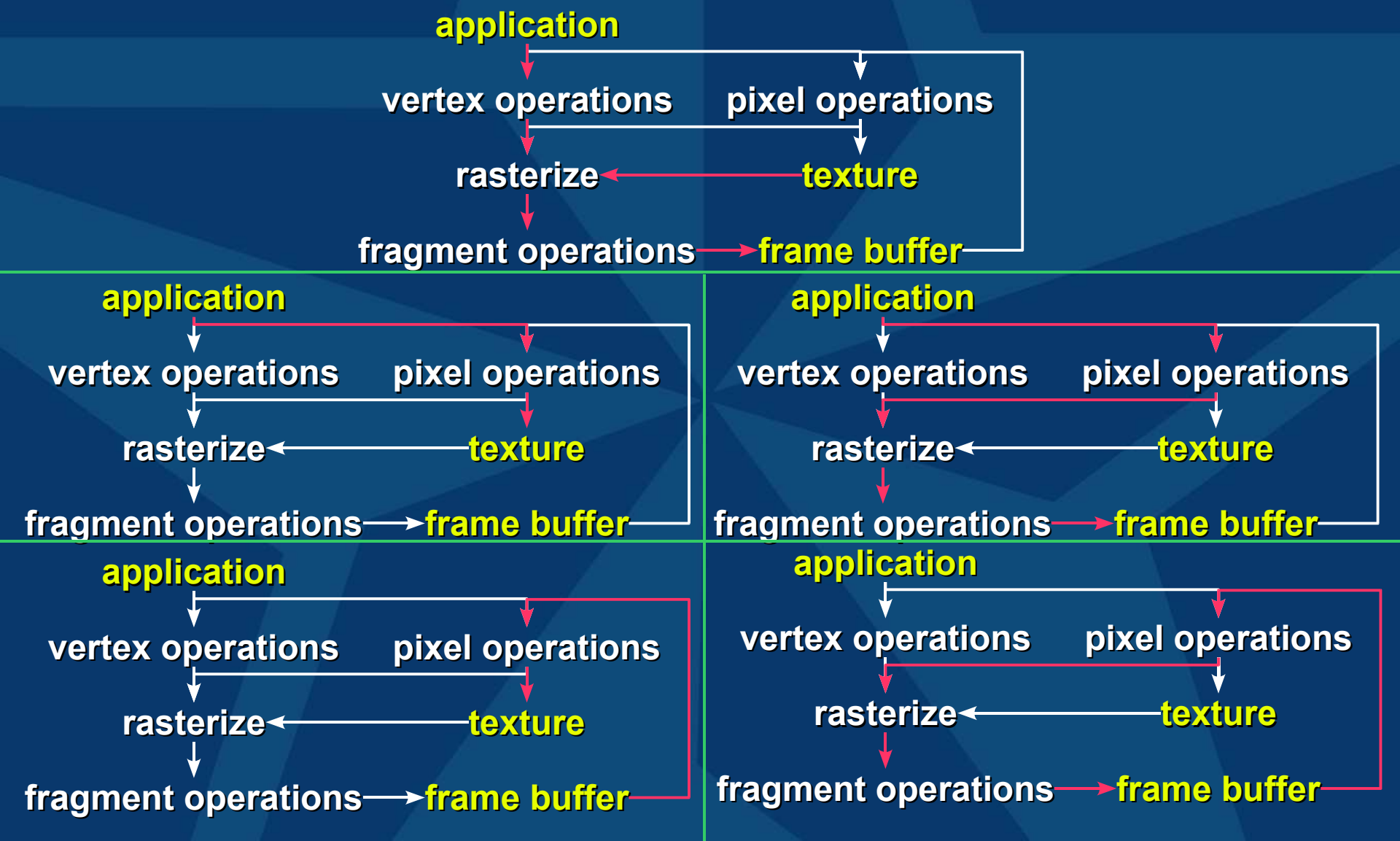
SAN ANTONIO SIGGRAPH 2002

Graphics hardware

- (as seen through OpenGL)



Rendering Passes



Multi-Pass = SIMD

Single Instruction, Multiple Data

Classic SIMD

- Thousands/millions of processors
- Thinking Machines, PixelFlow, …
- Not small-scale SIMD (MMX, etc.)

Shading languages use SIMD model

- Describe shading for one point
- Apply for every point on surface

Multi-Pass = SIMD

General SIMD	OpenGL
Shared Control	Application
Processor Array	Pixel Array
Per-PE ALU	Fragment ops
Per-PE Memory	FB / Texture
Per-PE Conditionals	Alpha / Stencil

What's it Mean?

We can create a compiler

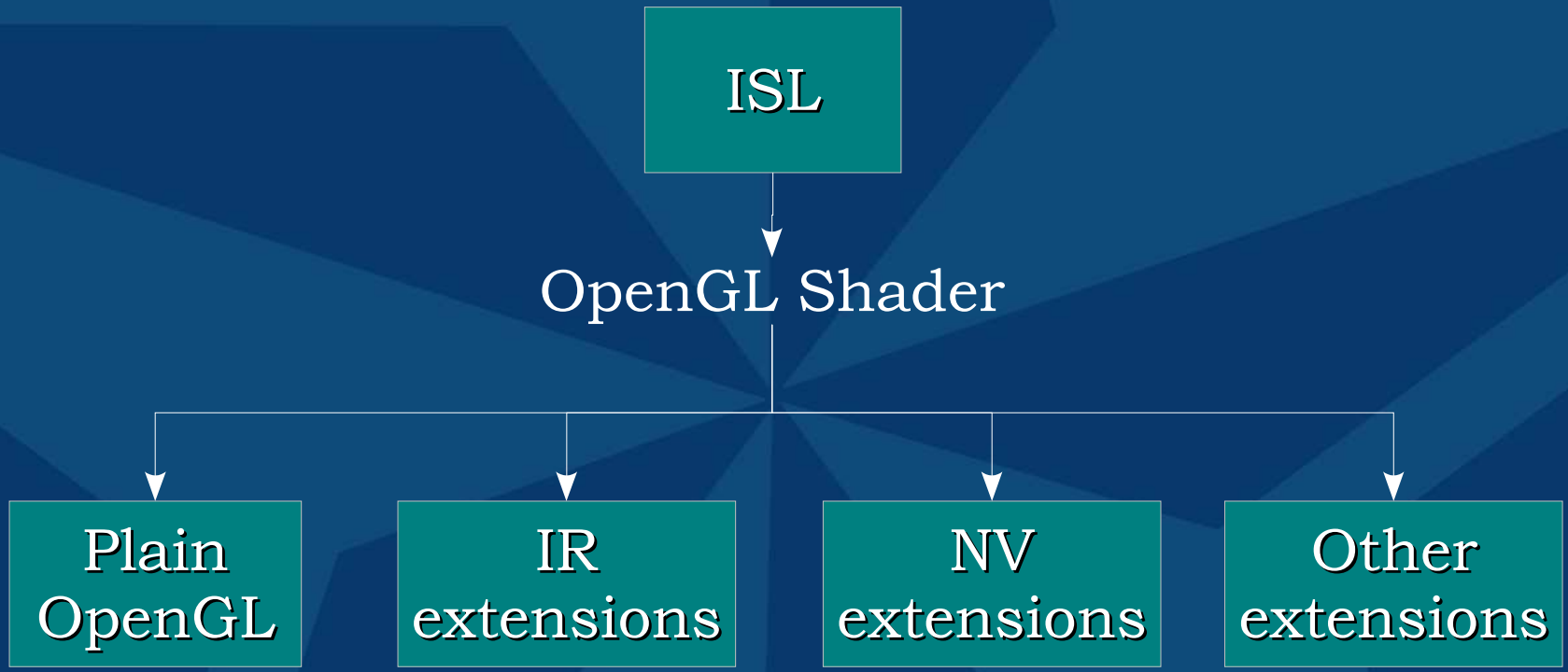
- High-level language in
- OpenGL out

Isn't that Slow?

No!

- Like drawing a few extra objects
- Optimize to compress passes
- Target hardware extensions

OpenGL Shader



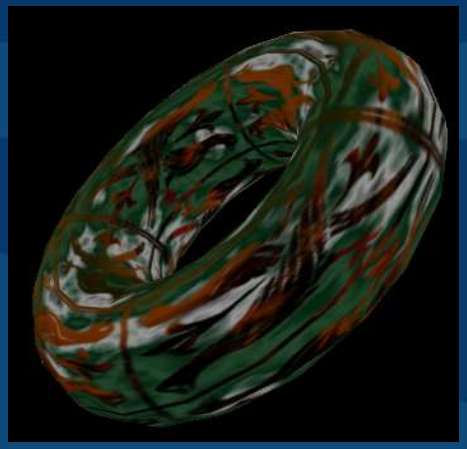
About ISL

Things exposed in ISL

- Pass count: passes <= statements
  - Optimize to fewer
- Range: clamped 0 – 1
- Texturing limits
  - No per-pixel computed texture coordinates
  - Can use per-vertex texture coordinates

I Can do that Myself!

SAN ANTONIO SIGGRAPH 2002



```
surface fancy( )
{
        FB = environment("flowers.rgb");
        FB *= color(.5,.2,.0,0);
        FB = under(texture("birds.rgba",
                scale(2,2,2)));
}
```

```
/* Draw Shader: fancy */
GLint draw_fancy_shader (void
(*draw_geometry) (void*),
                         void *geometry,
                         GLsizei win_w,
GLsizei win_h,
                         GLint x_lower_left,
GLint y_lower_left,
                         GLint x_upper_right,
GLint y_upper_right)
{
    GLsizei bounding_rect_w,
bounding_rect_h;
    GLenum light_id;
    GLfloat light_position[4];
    GLfloat light_ambient[4];
    GLfloat light_diffuse[4];
    GLfloat light_specular[4];
    GLfloat mat_emission[4];
    GLfloat mat_ambient[4];
    GLfloat mat_diffuse[4];
    GLfloat mat_specular[4];
    GLfloat curr_matrix[16];
    GLfloat texgen_plane[4];
    GLfloat tex_w_scale, tex_h_scale;

    if (setup_done == GL_FALSE) {
        return 0;
    }

    /* Error check bounding box x coords */
    if ((bounding_rect_w = x_upper_right -
x_lower_left + 1) &lt;= 0) {
        return -1;
    }

    /* Error check bounding box y coords */
    if ((bounding_rect_h = y_upper_right -
y_lower_left + 1) &lt;= 0) {
        return -1;
    }
    /* Error check window dimensions */
    if ((win_w &lt;= 0) || (win_h &lt;= 0))
{
        return -1;
    }

    /* Error check temporary texture
dimensions */
    if ((tmp_tex_w &lt;= 0) || (tmp_tex_h
&lt;= 0)) {
        return -1;
    }

    tex_w_scale = (GLfloat) win_w /
(GLfloat) tmp_tex_w;
    tex_h_scale = (GLfloat) win_h /
(GLfloat) tmp_tex_h;

#if 1

    /**********************************
     * Start Geom Pass.  Pass: 0
     **********************************/

    /* Save Default State */
    glPushAttrib (GL_ALL_ATTRIB_BITS);
    glMatrixMode (GL_TEXTURE);
    glPushMatrix ();
    glMatrixMode (GL_COLOR);
    glPushMatrix ();

    light_id = GL_LIGHT0;

    /* IPF_Colormask_Attribute */
    glColorMask (1, 1, 1, 1);

    /* IPF_Stencil_Attribute */
    glEnable (GL_STENCIL_TEST);
    glStencilFunc (GL_ALWAYS, 0x1, 0xff);
    glStencilOp (GL_KEEP, GL_KEEP,
GL_REPLACE);
    glStencilMask (0xff);

    /* IPF_Depth_Attribute */
    glEnable (GL_DEPTH_TEST);
    glDepthFunc (GL_LEQUAL);

    /* IPF_Texture_Attribute */
    glEnable (GL_TEXTURE_2D);
    glBindTexture (GL_TEXTURE_2D,
texture_ids[0]);  /* flowers.rgb */

    /* IPF_Texgen_Attribute */
    glMatrixMode (GL_MODELVIEW);
    glPushMatrix ();
    glLoadIdentity ();

    glEnable (GL_TEXTURE_GEN_S);
    glTexGeni (GL_S, GL_TEXTURE_GEN_MODE,
GL_SPHERE_MAP);

    glEnable (GL_TEXTURE_GEN_T);
    glTexGeni (GL_T, GL_TEXTURE_GEN_MODE,
GL_SPHERE_MAP);

    glPopMatrix ();

    /* Draw Geometry */
    /* Always set the matrix mode to
modelview.
       This could be a problem if the
callback
       expects the mode to be something
else. */
    glMatrixMode (GL_MODELVIEW);
    draw_geometry (geometry);

    /* Restore Default State */
    glMatrixMode (GL_TEXTURE);
    glPopMatrix ();
    glMatrixMode (GL_COLOR);
    glPopMatrix ();
    glPopAttrib ();

#endif

#if 1

    /**********************************
     * Start Geom Pass.  Pass: 1
     **********************************/

    /* Save Default State */
    glPushAttrib (GL_ALL_ATTRIB_BITS);
    glMatrixMode (GL_TEXTURE);
    glPushMatrix ();
    glMatrixMode (GL_COLOR);
    glPushMatrix ();

    light_id = GL_LIGHT0;

    /* IPF_Colormask_Attribute */
    glColorMask (1, 1, 1, 1);

    /* IPF_Stencil_Attribute */
    glEnable (GL_STENCIL_TEST);
    glStencilFunc (GL_LEQUAL, 0x1, 0xff);
    glStencilOp (GL_KEEP, GL_KEEP,
GL_KEEP);
    glStencilMask (0x7fffffff);

    /* IPF_Blend_Attribute */
    glEnable (GL_BLEND);
    glBlendEquationEXT (GL_FUNC_ADD_EXT);
    glBlendFunc (GL_DST_COLOR, GL_ZERO);

    /* IPF_Depth_Attribute */
    glEnable (GL_DEPTH_TEST);
    glDepthFunc (GL_LEQUAL);

    /* IPF_Color_Attribute */
    glColor4f (0.500000, 0.200000,
0.000000, 0.000000);

    /* Draw Geometry */
    /* Always set the matrix mode to
modelview.
       This could be a problem if the
callback
       expects the mode to be something
else. */
    glMatrixMode (GL_MODELVIEW);
    draw_geometry (geometry);

    /* Restore Default State */
    glMatrixMode (GL_TEXTURE);
    glPopMatrix ();
    glMatrixMode (GL_COLOR);
    glPopMatrix ();
    glPopAttrib ();

#endif

#if 1

    /**********************************
     * Start Geom Pass.  Pass: 2
     **********************************/

    /* Save Default State */
    glPushAttrib (GL_ALL_ATTRIB_BITS);
    glMatrixMode (GL_TEXTURE);
    glPushMatrix ();
    glMatrixMode (GL_COLOR);
    glPushMatrix ();

    light_id = GL_LIGHT0;

    /* IPF_Colormask_Attribute */
    glColorMask (1, 1, 1, 1);

    /* IPF_Stencil_Attribute */
    glEnable (GL_STENCIL_TEST);
    glStencilFunc (GL_LEQUAL, 0x1, 0xff);
    glStencilOp (GL_KEEP, GL_KEEP,
GL_KEEP);
    glStencilMask (0x7fffffff);

    /* IPF_Blend_Attribute */
    glEnable (GL_BLEND);
    glBlendEquationEXT (GL_FUNC_ADD_EXT);
    glBlendFunc (GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA);

    /* IPF_Depth_Attribute */
    glEnable (GL_DEPTH_TEST);
    glDepthFunc (GL_LEQUAL);

    /* IPF_Texture_Attribute */
    glEnable (GL_TEXTURE_2D);
    glBindTexture (GL_TEXTURE_2D,
texture_ids[1]);  /* marblebirds.rgba */

    /* IPF_Matrix_Attribute */
    glMatrixMode (GL_TEXTURE);
    curr_matrix[0] = 2.000000;
    curr_matrix[1] = 0.000000;
    curr_matrix[2] = 0.000000;
    curr_matrix[3] = 0.000000;
    curr_matrix[4] = 0.000000;
    curr_matrix[5] = 2.000000;
    curr_matrix[6] = 0.000000;
    curr_matrix[7] = 0.000000;
    curr_matrix[8] = 0.000000;
    curr_matrix[9] = 0.000000;
    curr_matrix[10] = 2.000000;
    curr_matrix[11] = 0.000000;
    curr_matrix[12] = 0.000000;
    curr_matrix[13] = 0.000000;
    curr_matrix[14] = 0.000000;
    curr_matrix[15] = 1.000000;
    glLoadMatrixf (curr_matrix);

    /* Draw Geometry */
    /* Always set the matrix mode to
modelview.
       This could be a problem if the
callback
       expects the mode to be something
else. */
    glMatrixMode (GL_MODELVIEW);
    draw_geometry (geometry);

    /* Restore Default State */
    glMatrixMode (GL_TEXTURE);
    glPopMatrix ();
    glMatrixMode (GL_COLOR);
    glPopMatrix ();
    glPopAttrib ();

#endif

    return 0;
}
```

Shading Example

```
#include <swizzle.h>
uniform color greentable[2] =
{color(0,.2,0,1), color(0,.4,0,1)};

surface
toon( parameter float do_toon = 1.;
      parameter float edge = .25)
{
  FB = environment("redpark.env");
  if (do_toon > .5) {
      FB += edge;
      FB = transform(rgba_rrra);
      FB = lookup(greentable);
      FB += environment("sun.env");
  }
}
```

1: Uniform Expressions


```
#include <swizzle.h>
uniform color greentable[2] =
{color(0,.2,0,1), color(0,.4,0,1)};

surface
toon( uniform float do_toon = 1.;
      uniform float edge = .25)
{
  FB = environment("redpark.env");
  if (do_toon > .5) {
      FB += edge;
      FB = transform(rgba_rrra);
      FB = lookup(greentable);
      FB += environment("sun.env");
  }
}
```

1: Simple Operations


```
FB = environment(redpark)
```

```
FB += .25
```

```
FB = transform(rgba_rrra)
```

```
FB = lookup(greentable)
```

```
FB += environment(sun)
```

```
a = texgen(environment)
b = lookup(redpark,a)
```

```
c = const(.25)
d = add(b,c)
```

```
e = transform(rgba_rrra,d)
```

```
f = lookup(greentable,e)
```

```
g = texgen(environment)
h = lookup(sun,g)
i = add(f,h)
```

1: Map to Hardware

Pack passes back-to-front

Remove dead code

```
a = texgen(environment)
b = lookup(redpark,a)
c = const(.25)
d = add(b,c)
e = transform(rgba_rrra,d)
f = lookup(greentable,e)
g = texgen(environment)
h = lookup(sun,g)
i = add(f,h)
```

```
a = texgen(environment)
b = lookup(redpark,a)
c = const(.25)
d = add(b,c)
e = transform(rgba_rrra,d)
f = lookup(greentable,e)
g = texgen(environment)
h = lookup(sun,g)
i = add(f,lookup(sun,
    texgen(environment)))
```

1: Map to Hardware

Pack passes back-to-front

Remove dead code

```
a = texgen(environment)
b = lookup(redpark,a)
c = const(.25)
d = add(b,c)
e = transform(rgba_rrra,d)
f = lookup(greentable,e)
g = texgen(environment)
h = lookup(sun,g)
i = add(f,h)
```

```
a = texgen(environment)
b = lookup(redpark,a)
c = const(.25)
d = add(b,c)
e = transform(rgba_rrra,d)
f = lookup(greentable,
    transform(rgba_rrra,d))

i = add(f,lookup(sun,
    texgen(environment)))
```

1: Map to Hardware

Pack passes back-to-front

Remove dead code

```
a = texgen(environment)
b = lookup(redpark,a)
c = const(.25)
d = add(b,c)
e = transform(rgba_rrra,d)
f = lookup(greentable,e)
g = texgen(environment)
h = lookup(sun,g)
i = add(f,h)
```

```
a = texgen(environment)
b = lookup(redpark,a)
c = const(.25)
d = add(b,const(.25))

f = lookup(greentable,
    transform(rgba_rrra,d))

i = add(f,lookup(sun,
    texgen(environment)))
```

1: Map to Hardware

Pack passes back-to-front

Remove dead code

```
a = texgen(environment)
b = lookup(redpark,a)
c = const(.25)
d = add(b,c)
e = transform(rgba_rrra,d)
f = lookup(greentable,e)
g = texgen(environment)
h = lookup(sun,g)
i = add(f,h)
```

```
a = texgen(environment)
b = lookup(redpark,
    texgen(environment))
d = add(b,const(.25))

f = lookup(greentable,
    transform(rgba_rrra,d))

i = add(f,lookup(sun,
    texgen(environment)))
```

1: Map to Hardware

Pack passes back-to-front

Remove dead code

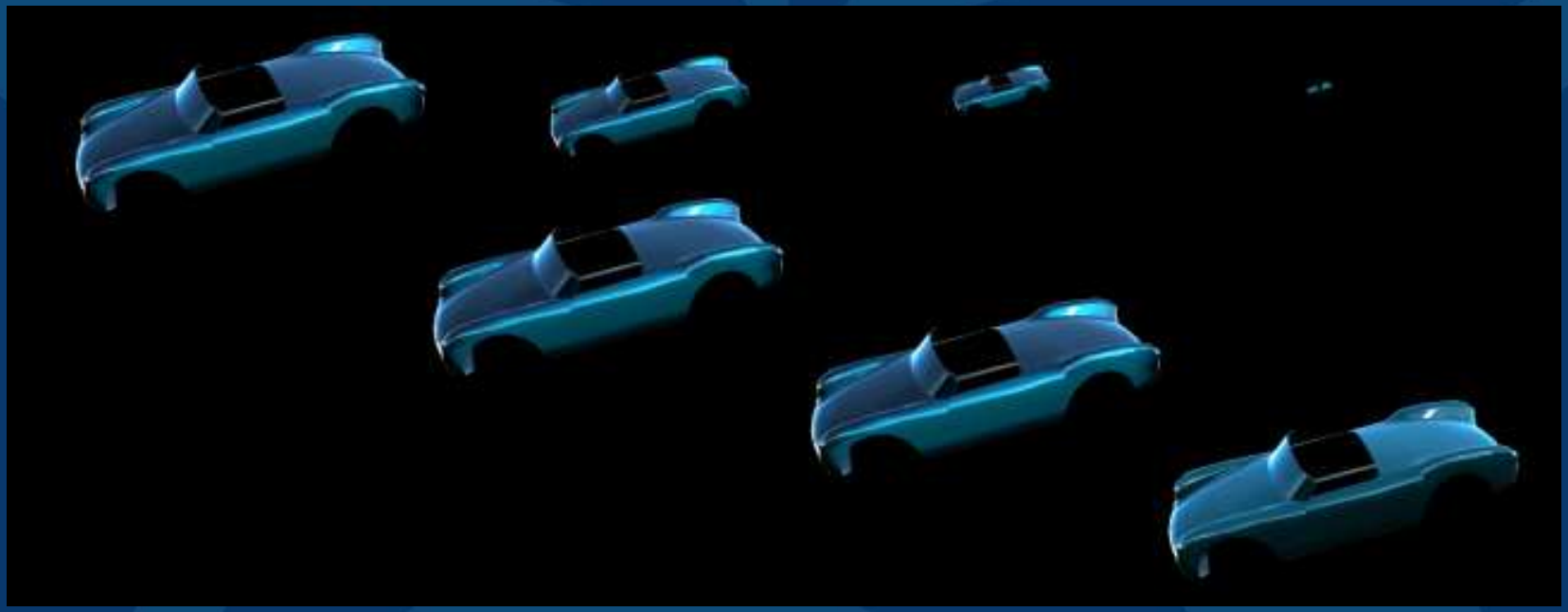
```
a = texgen(environment)
b = lookup(redpark,a)
c = const(.25)
d = add(b,c)
e = transform(rgba_rrra,d)
f = lookup(greentable,e)
g = texgen(environment)
h = lookup(sun,g)
i = add(f,h)
```

```
b = lookup(redpark,
    texgen(environment))
d = add(b,const(.25))

f = lookup(greentable,
    transform(rgba_rrra,d))

i = add(f,lookup(sun,
    texgen(environment)))
```

Level-of-detail Shaders

Add conditionals to adjust complexity

- Distance
- Importance
- Time
- Available texture



Level-of-detail

Automatic

- Add conditionals
- Change “hardware mapping” rules in each branch

Semi-automatic

- Use LOD building blocks

Manual

- Add conditionals
- Hand-code levels

Wrap-up

Simple, easy-to-change description

- Flexible surface appearance
- Much easier than hand-coded GL

Portable

Enables powerful extra processing

SAN ANTONIO

SIGGRAPH 2002