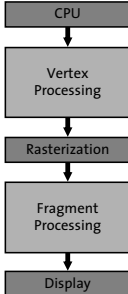


# Graphics Pipeline & APIs




```

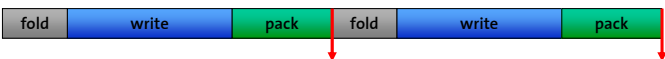
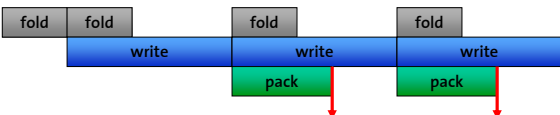
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glPushMatrix ();
glTranslatef (-0.15, -0.15, solidZ);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_zero);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_solid);
glCallList (sphereList);
glPopMatrix ();

```




# Pipelines

- One person prepares Xmas cards:
 
- Three persons prepare Xmas cars:
 

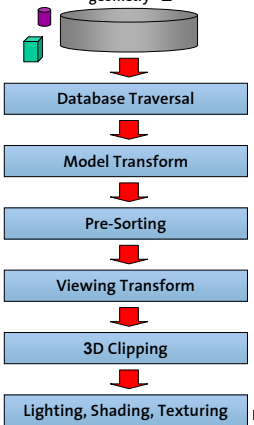
- Ideal:  $n$  stages  $\rightarrow$  speedup  $n$
- Bottleneck: slowest stage
- Graphics: Bottleneck determines frames/s

2  
1. Graphics APIs



# The Graphics Pipeline

attributed geometry -  $\Delta$



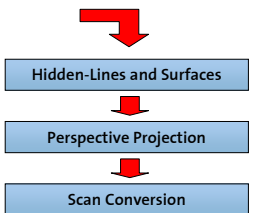
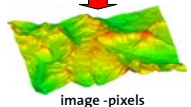




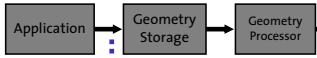
image - pixels

3  
1. Graphics APIs




# The Graphics Pipeline

Vertices



CPU

Pixels



GPU

4  
The Graphics Pipeline



## Application

- Generate database
  - Usually only once
  - Load from disk
  - Build acceleration structures (hierarchies,...)
- Database traversal
- Input event handlers
- Modify data structures
- Simulation
- Primitive generation (e.g. triangles)
- other functions..

5

The Graphics Pipeline



## Geometry Storage

- Command buffering
- Command interpretation
- Unpack and perform format conversion
- Maintain graphics state

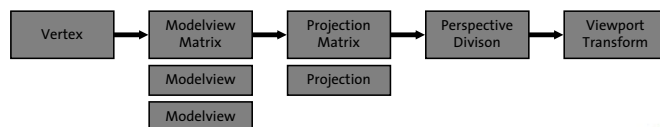
6

The Graphics Pipeline



## Geometry Processor

- Evaluation of polynomials for curved surfaces
- Transformation and projection



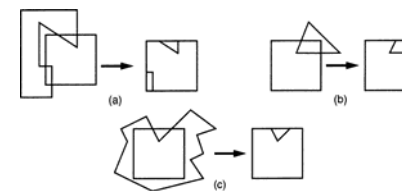
7

The Graphics Pipeline



## Geometry Processor (2)

- Evaluation of polynomials for curved surfaces
- Transformation and projection
- Clipping, culling, and primitive assembly



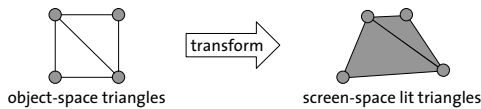
8

The Graphics Pipeline



## Geometry Processor (3)

- Evaluation of polynomials for curved surfaces
- Transformation and projection
- Clipping, culling, and primitive assembly
- Lighting
- Texture coordinate generation



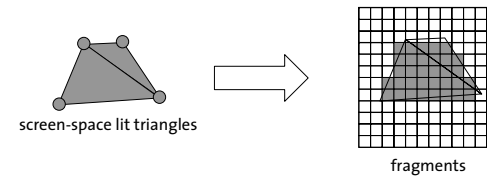
9

The Graphics Pipeline



## Rasterization

- Setup (per-triangle)
- Sampling (triangle = {fragments})
- Interpolation (interpolate colors and coordinates)



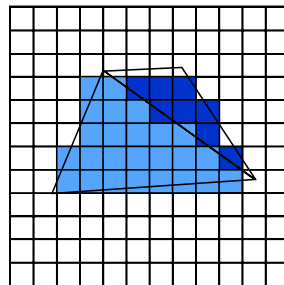
10

The Graphics Pipeline



## Rasterization (2)

- Separate rule for each primitive
- Non-ambiguous!
- Polygons:
  - Pixel center contained in polygon
  - On-edge pixels only one is rasterized



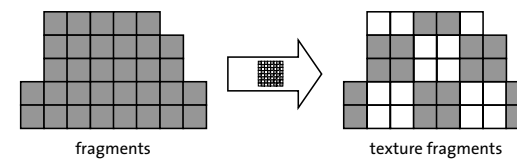
11

The Graphics Pipeline



## Texture

- Texture transformation and projection
- Texture address calculation
- Texture filtering



12

The Graphics Pipeline

**Texture (2)**

- Texture combiners

The diagram illustrates the texture combining process. On the left, there are two separate grids: 'texture fragments' (a 4x4 grid of grey and white squares) and 'fragments' (a 4x4 grid of red squares). An arrow points to the right, where the two are combined into 'textured fragments', which is a 4x4 grid where the red squares from the original fragments are now overlaid on the grey and white squares.

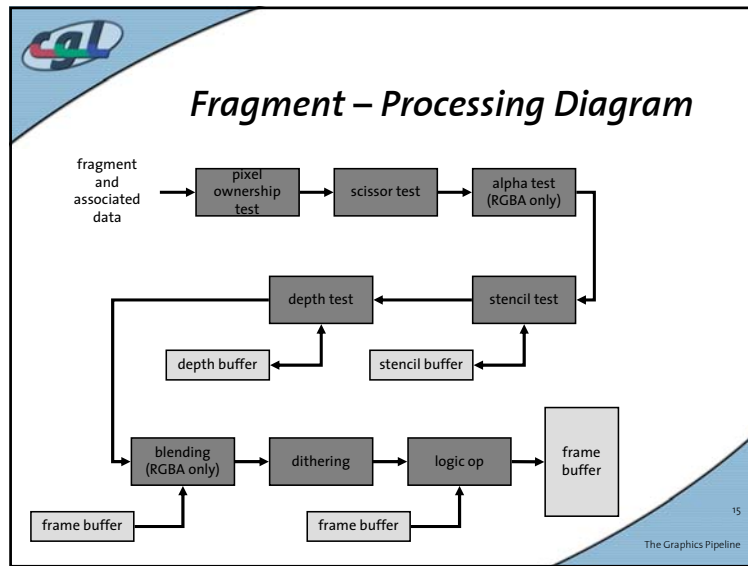
13  
The Graphics Pipeline

**Fragment - Processing**

- Texture combiners and fog
- Owner, scissor, depth, alpha, stencil tests
- Blending or compositing
- Dithering and logical operations

The diagram shows the transition from 'textured fragments' (a 4x4 grid of grey and red squares) to 'Framebuffer pixels' (a larger 10x10 grid). An arrow points from the textured fragments to the framebuffer pixels, which now contain a detailed, shaded version of the original texture.

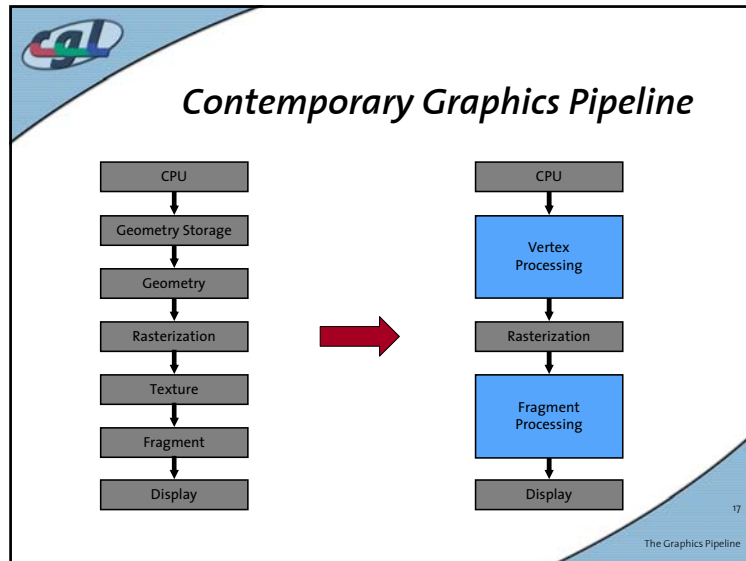
14  
The Graphics Pipeline



**Display**

- Frame buffer pixel format
  - RGBA vs. indexed color
- Bits: 16-bit, 32-bit, 128-bit floating point
- Double buffer vs. single buffer
- Quad-buffer stereo
- Overlays (extra bit planes)
- Auxiliary buffers: alpha, stencil

16  
The Graphics Pipeline



- 
- Contemporary Graphics Pipeline (2)**
- Vertex Processing = per-vertex
    - Transform & Lighting (T&L)
    - Historically: hardwired complex operations (floating point)
    - Today: programmable
    - flow control, texture lookup
    - 400 million vertices per second
  - Fragment Processing = per-fragment
    - Blending and texture combination
    - Historically: fixed point and limited operations
    - Today: programmable
    - 4 billion fragments ("Gigapixel") per second
    - New: floating point, complex operations
- 18  
The Graphics Pipeline

**Graphics Application Programming Interfaces (APIs)**

```

glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glPushMatrix ();
glTranslatef (-0.15, -0.15, solidZ);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_zero);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_solid);
glCallList (sphereList);
glPopMatrix ();
  
```

Graphics APIs

- 
- Graphics Libraries (API's)**
- Give access to graphics hardware
  - Declarative (what, not how)
    - Describe the scene
    - Scene Graph API (**retained mode**)
    - e.g. SGI Open Inventor, SGI Performer, Renderman
  - Imperative (how, not what)
    - Sequence of drawing commands
    - **Immediate mode** API
    - e.g. OpenGL, DirectX (D3D), Postscript
    - More direct control
- 20  
Graphics APIs



## OpenGL (Open Graphics Library)

- Initially defined by Silicon Graphics Inc.
  - <http://www.opengl.org/>
  - <http://www.sgi.com/software/opengl/>
- OpenGL Architectural Review Board (ARB)
  - 3DLabs, Apple, ATI, Compaq, Evans&Sutherland, hp, IBM, Intel, Microsoft, nVidia, sgi, SUN
- Available on many platforms
  - Windows NT/v4, Windows XP/2000/98/95
  - various UNIX (IRIX, Solaris, etc.)
  - Linux (Debian, RedHat, SuSE, Caldera)
  - Mac OS 8/9/X

21

OpenGL



## OpenGL (2)

- Scales from PC to image engines
- Intuitive, procedural interface
- Versions
  - 1.4 widespread
  - 2.0 available since August 2004
- Language bindings
  - C, Java, Ada, Fortran, Perl, Python

22

OpenGL



## A 2D OpenGL Example

```
#include <GL/gl.h>
#include <GL/glu.h>

main() {
    OpenAWindow();
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
    glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0); glVertex2f( 0.9, -0.9);
        glColor3f(0.0, 1.0, 0.0); glVertex2f( 0.0,  0.9);
        glColor3f(0.0, 0.0, 1.0); glVertex2f(-0.9, -0.9);
    glEnd();
    glFlush();
    SomeMainLoop();
}
```



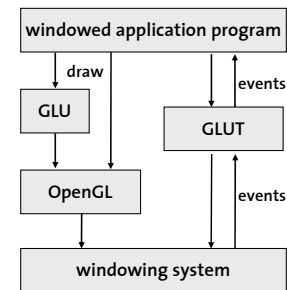
23

OpenGL



## Integration

- gl.lib:** main library
  - supports NURBS,
  - complex polygons,
  - quadric shapes, etc.
- glut.lib:** toolkit
  - simplifies & abstracts window handling



24

OpenGL



## Basics

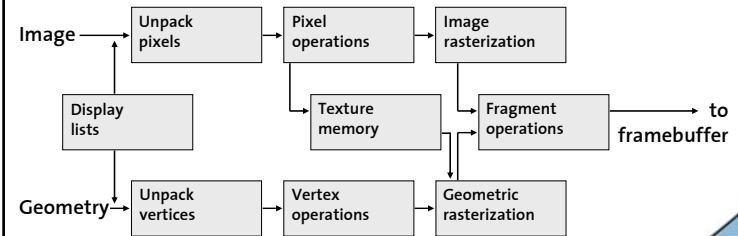
- OpenGL is a state machine
  - State encapsulates control for lighting, shading, texturing, etc.
  - Current state affects transforms, color, lighting, shading, etc.
- Vertices and pixels are fundamental primitives
- Display lists to optimize performance
  - `glNewList(<id>, GL_COMPILE);`
- Caching of graphics commands

25

OpenGL



## Data Flow



[ fragment = pixel ]

26

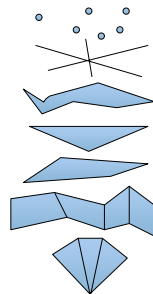
OpenGL



## Geometric Primitives

```

glBegin(X);
X: GL_POINTS,
    GL_LINES,
    GL_POLYGON,
    GL_TRIANGLES,
    GL_QUADS,
    GL_TRIANGLE_STRIP,
    GL_TRIANGLE_FAN,
glEnd();
  
```



27

OpenGL



## Rendering Features

- Materials & colors
- Texture mapping
- Texture animation
- MipMaps
- Advanced shading and lighting
- Fog
- Antialiasing
- Framebuffer Ops



28

OpenGL



## A 3D OpenGL Example

```
glClearColor(0.0, 0.0, 0.0, 0.0);
GLfloat gray[] = {0.5, 0.5, 0.5, 1.0};
GLfloat white[] = {1.0, 1.0, 1.0, 1.0};
GLfloat ldir[] = {1.0, 1.0, 1.0, 0.0};
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_AMBIENT, gray);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_POSITION, ldir);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_DEPTH_TEST);
```

*Lighting*



```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(40, 1.0, 0.1, 10); // fovy, aspect, near, far
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0,0.0,2.0, 0.0,0.0,0.0, 0.0,1.0,0.0); // eye, center, up
```

*Transforms*

```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glColor3f(0.5, 0.3, 0.3);
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glRotatef(30, 1.0, 1.0, 1.0);
glutSolidTeapot(0.5);
glPopMatrix();
```

*Geometry*

29

OpenGL



## OpenGL Extensions

- Enables access to hardware-specific features
- Specified by 3D video card manufacturers & OpenGL vendors
- Extensions defined in [glext.h](http://www.opengl.org/registry/doc/glext.h)
- Extensions have manufacturer's postfix, e.g.
  - ARB = official extensions by OpenGL Architectural Review Board
  - EXT = agreed upon by multiple OpenGL vendors
  - SGI = Silicon Graphics
  - NV = nVidia Corporation
- Examples:
  - `glEnable(GL_VERTEX_PROGRAM_NV)`
  - `glLoadProgramNV()`

30

OpenGL