Advanced OpenGL Topics

Jian Huang, CS456

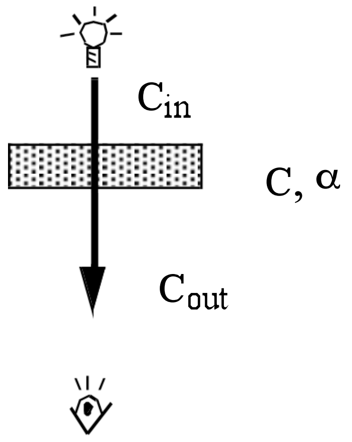
Alpha: the 4th Color Component

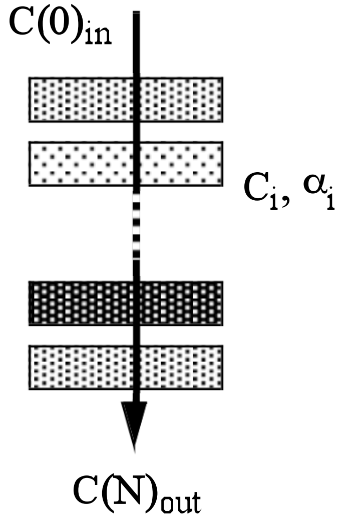
- Measure of Opacity
  - simulate translucent objects
    - glass, water, etc.
  - composite images
  - antialiasing
  - ignored if blending is not enabled

```
glEnable( GL_BLEND )
```

Compositing for Semi-transparency

- Requires sorting! (BTF vs. FTB)
- "over" operator - Porter & Duff 1984
- α : opacity

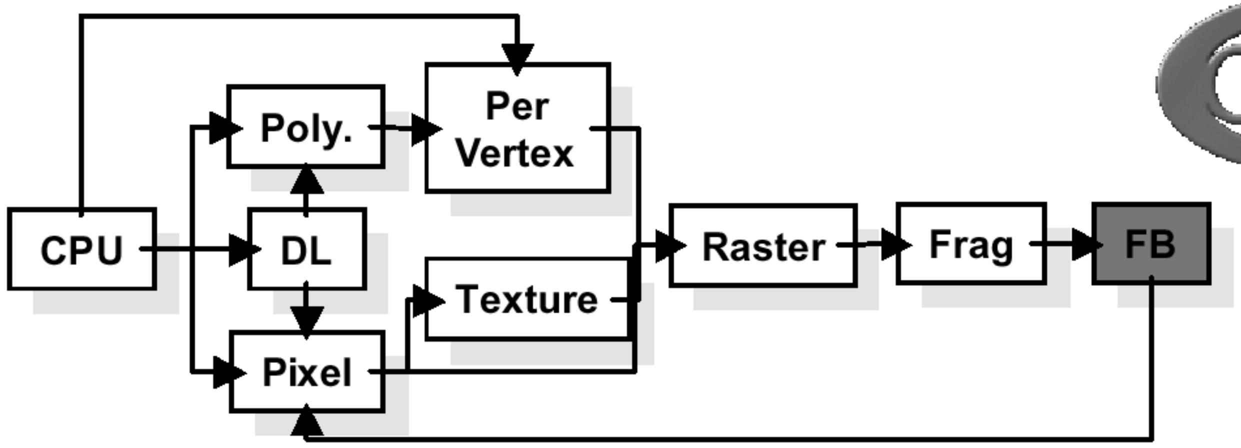




$$C_{out} = C_{in} \cdot (1-\alpha) + C \cdot \alpha \qquad C(i)_{in} = C(i-1)_{out}$$

Fragment

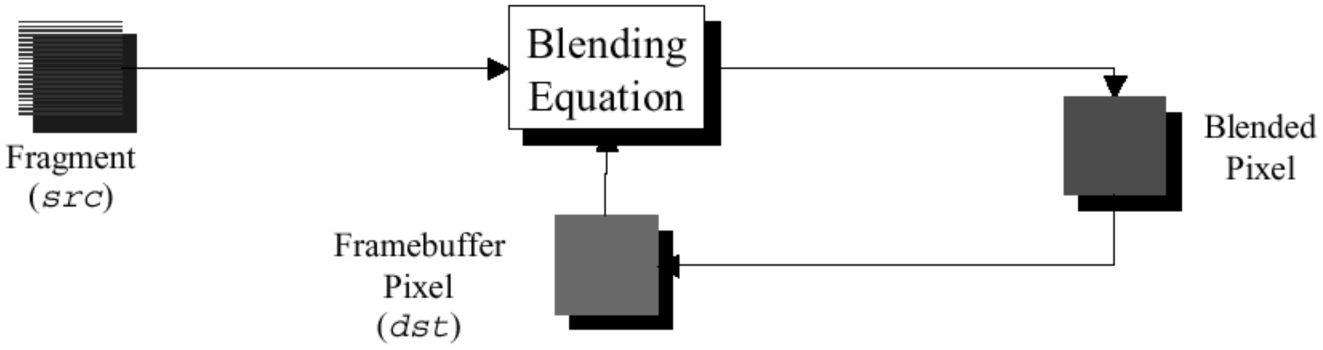
- Fragment in OGL: after the rasterization stage (including texturing), the data are not yet pixel, but are fragments
  - Fragment is all the data associated with a pixel, including coordinate, color, depth and texture coordinates.



Blending in OGL

- If a fragment makes it to FB, the pixel is read out and blended with the fragment's color and then written back to FB
- The contributions of fragment and FB pixel is specified: **glBlendFunc(src, dst)**

$$\bar{C}_r = src\ \bar{C}_f + dst\ \bar{C}_p$$



Blending Function

- Four choices:
  - GL_ONE
  - GL_ZERO
  - GL_SRC_ALPHA
  - GL_ONE_MINUS_SRC_ALPHA
- Blending is enabled using glEnable(GL_BLEND)
- *Note:* If your OpenGL implementation supports the GL_ARB_imaging extension, you can modify the blending equation as well.

3D Blending with Depth Buffer

- A scene of opaque and translucent objects
  - Enable depth buffering and depth test
  - Draw opaque objects
  - Make the depth buffer read only, with glDepthMask(GL_FALSE)
  - Draw the translucent objects (sort those triangles still, but which order, FTB or BTF?)

How Many Buffers Are There?

- In OpenGL:
  - Color buffers: front, back, front-left, front-right, back-left, back-right and other auxiliaries
  - Depth buffer
  - Stencil buffer
  - Accumulation buffer
- Exactly how many bits are there in each buffer, depends on
  - OGL implementation
  - what you choose
- Each buffer can be individually cleared

Selecting Color Buffer for Writing and Reading

- Drawing and reading can go into any of front, back, front-left, front-right, back-left, back-right and other auxiliaries
- glDrawBuffer: select buffer to be written or drawn into
  - Can have multiple writing buffer at the same time
- glReadBuffer: select buffer as the source for
  - glReadPixels
  - glCopyPixels
  - glCopyTexImage (copy from FB to a texture image)
  - glCopyTexSubImage

Accumulation Buffer

- Problems of compositing into color buffers
  - limited color resolution (e.g. 8 bits/channel)
    - clamping
    - loss of accuracy
- Accumulation buffer acts as a “floating point” color buffer
  - accumulate into accumulation buffer
  - transfer results to frame buffer

Accumulation Buffer

- OGL don’t directly write into A-buffer
- Typically, a series of images are rendered to a standard color buffer and then accumulated, one at a time, into the A-buffer.
- Then, the result in A-buffer has to be copied back to a color buffer for viewing
- A-buffer may have more bits/color

Accessing Accumulation Buffer

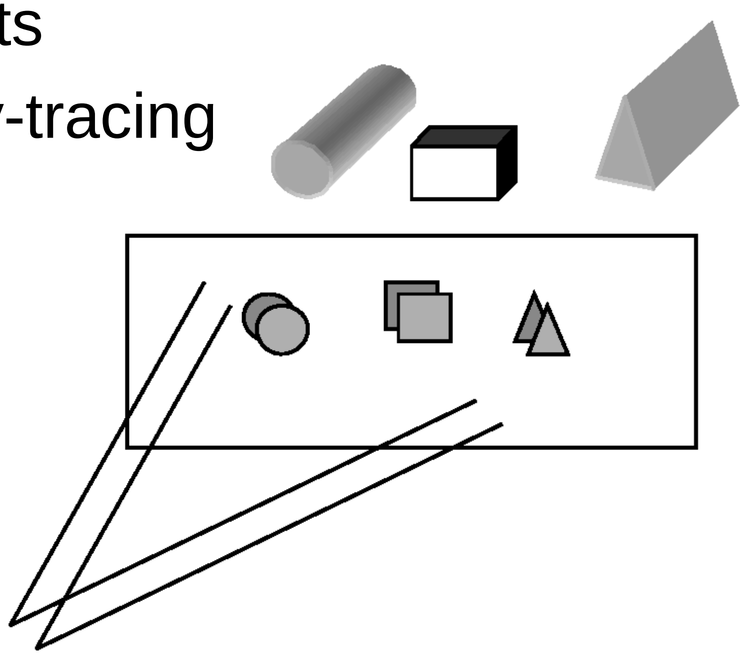
- **glAccum(op, value)** operations
  - within the accumulation buffer:
    - **GL_ADD, GL_MULT**
  - from read buffer
    - **GL_ACCUM, GL_LOAD**
  - transfer back to write buffer
    - **GL_RETURN**
  - **glAccum(GL_ACCUM, 0.5)** multiplies each value in read buffer by 0.5 and adds to accumulation buffer
  - How do you average N images?!

Accumulation Buffer Applications

- Compositing
- Full Scene Anti-aliasing
- Depth of Field
- Filtering
- Motion Blur

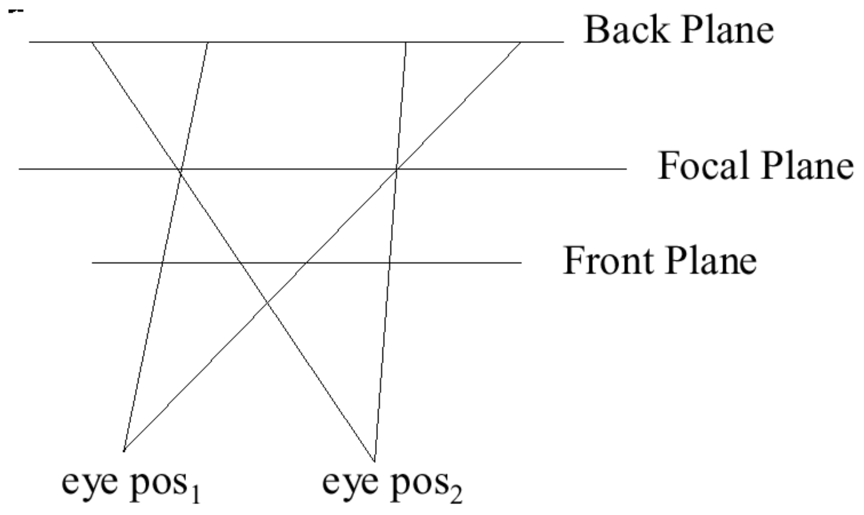
Full Scene Antialiasing

- Jittering the view (how??)
- Each time we move the viewer, the image shifts
- Different aliasing artifacts in each image
- Averaging images using accumulation buffer averages out these artifacts
- Like super-sampling in ray-tracing



Depth of Focus

- Keeping a Plane in Focus
- Jitter the viewer to keep one plane unchanged



How do you jitter the viewer ????

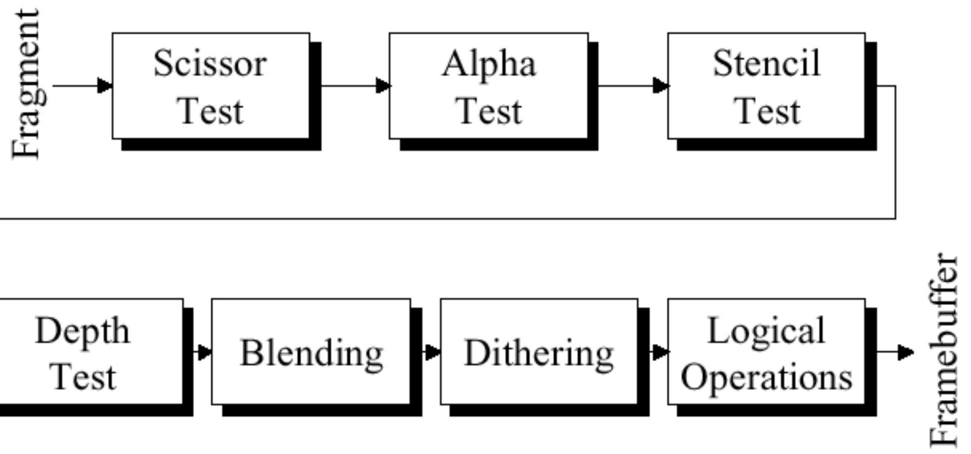
Move the camera in a plane parallel to the focal plane.

What else?

- Can you do soft shadows?
  - Jitter the light sources
- What about motion blur
  - Jitter the moving object in the scene
  - glAccum(GL_MULT, decayFactor)

A Fragment’s Journey Towards FB

- Many Tests (on/off with glEnable)
  - *scissor test* - an additional clipping test
  - *alpha test* - a filtering test based on alpha
  - *stencil test* - a pixel mask test
  - *depth test* - fragment occlusion test



Scissor Box

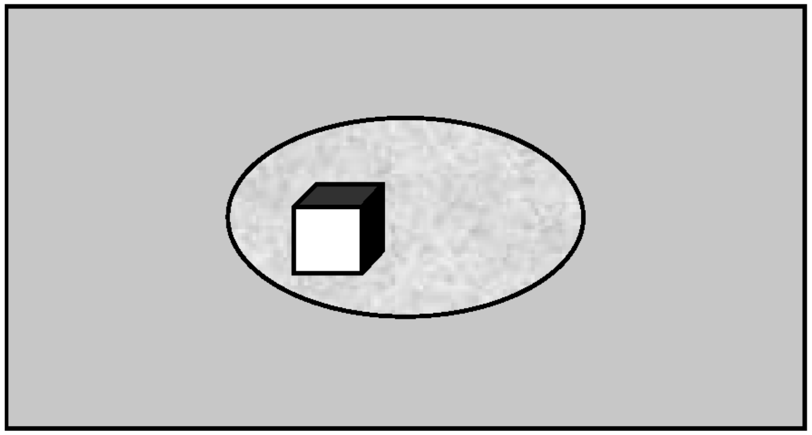
- Additional Clipping test
  - `glScissor( x, y, w, h )`
- any fragments outside of box are clipped
- useful for updating a small section of a viewport
- affects **`glClear()`** operations

Alpha Test

- Reject pixels based on their alpha value
- **`glAlphaFunc(func, value)`**
  - GL_NEVER GL_LESS
  - GL_EQUAL GL_LEQUAL
  - GL_GREATER GL_NOTEQUAL
  - GL_GEUQAL GL_ALWAYS
- For instance, use as a mask for texture
  - How would you render a fence?

Stencil Test

- Used to control drawing based on values in the stencil buffer
- Fragments that failt the stencil test are not drawn
  - Example: create a mask in stencil buffer and draw only objects not in mask area



Stencil Buffer

- Don’t render into stencil buffer
  - Control stencil buffer values with stencil function
  - Can change stencil buffer values with each fragment passing or failing the test

Controlling Stencil Buffer

- For each pixel, what do I do? Look:
  - **glStencilFunc(func, ref, mask)**
    - compare value in buffer with (**ref AND mask)** and **(stencil_pixel AND mask)** using **func**
    - **func** is one of standard comparison functions
  - **glStencilOp(fail, zfail, zpass)**
    - Allows changes in stencil buffer based on:
    - Failing stencil test
    - Failing depth test
    - Passing depth test
    - **GL_KEEP, GL_INCR, GL_REPLACE, GL_DECR, GL_INVERT, GL_ZERO**

Creating a Mask

- Initialize Mask

```
glInitDisplayMode( …|GLUT_STENCIL|… );
glEnable( GL_STENCIL_TEST );
glClearStencil( 0x0 );
glStencilFunc( GL_ALWAYS, 0x1, 0x1 );
glStencilOp( GL_REPLACE, GL_REPLACE,
GL_REPLACE );
```

Using Stencil Mask

- glStencilFunc(GL_EQUAL, 0x1, 0x1)

  draw objects where stencil = 1
- glStencilFunc(GL_NOT_EQUAL, 0x1, 0x1);
- glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

  draw objects where stencil != 1

Immediate Mode vs Display Lists

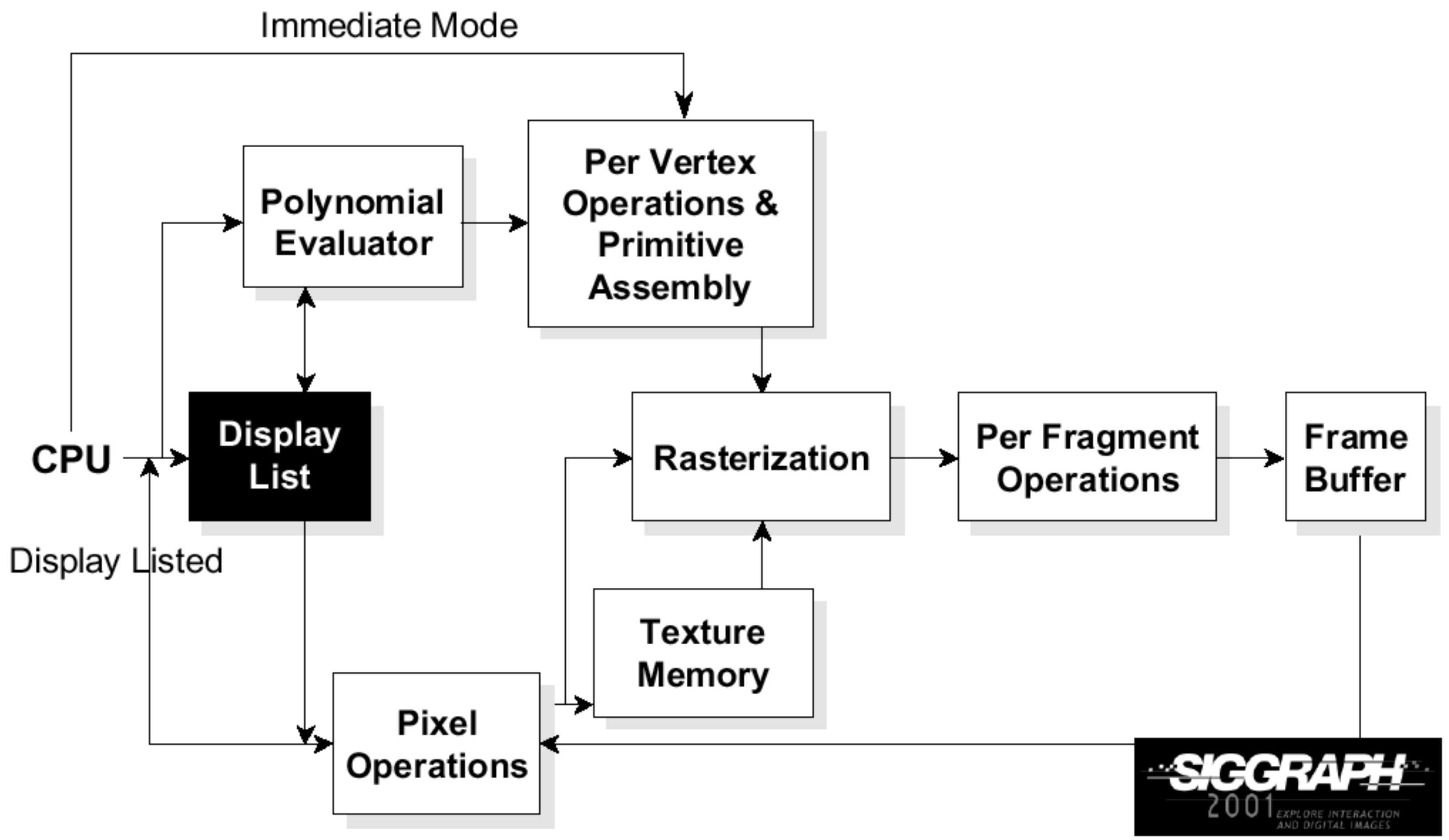
- Immediate Mode Graphics
  - Primitives are sent to pipeline and display right away
  - No memory of graphical entities
- Display Listed Graphics
  - Primitives placed in display lists
  - Display lists kept on graphics server
  - Can be redisplayed with different state
  - Can be shared among OpenGL graphics contexts (in X windows, use the `glXCreateContext()` routine)

Immediate Mode vs Retained Mode

- In immediate mode, primitives (vertices, pixels) flow through the system and produce images. These data are lost. New images are created by reexecuting the display function and regenerating the primitives.

- In retained mode, the primitives are stored in a display list (in “compiled” form). Images can be recreated by “executing” the display list. Even without a network between the server and client, display lists should be more efficient than repeated executions of the display function.

Immediate Mode vs Display Lists



SIGGRAPH 2001 EXPLORE INTERACTION AND DIGITAL IMAGES

Display Lists

Creating a display list

```
GLuint id;
void init( void )
{
    id = glGenLists( 1 );
    glNewList( id, GL_COMPILE );
    /* other OpenGL routines */
    glEndList();
}
```

Call a created list

```
void display( void )
{
    glCallList( id );
}
```

Instead of `GL_COMPILE`, `glNewList` also accepts constant `GL_COMPILE_AND_EXECUE`, which both creates and executes a display list.

If a new list is created with the same identifying number as an existing display list, the old list is replaced with the new calls. No error occurs.

Display Lists

- Not all OpenGL routines can be stored in display lists
  - If there is an attempt to store any of these routines in a display list, the routine is executed in immediate mode. No error occurs.
- State changes persist, even after a display list is finished
- Display lists can call other display lists
- Display lists are not editable, but can fake it
  - make a list (A) which calls other lists (B, C, and D)
  - delete and replace B, C, and D, as needed

Some Routines That Cannot be Stored in a Display List

Some routines cannot be stored in a display list. Here are some of them:

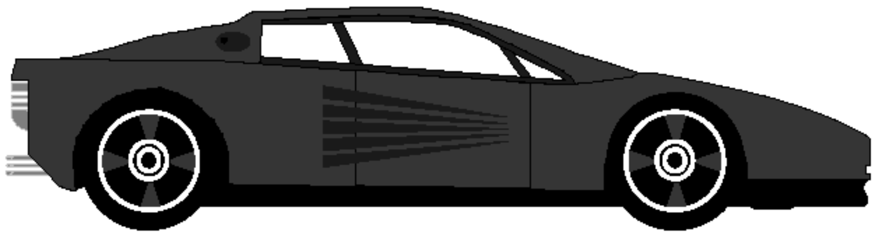
all glGet* routines

glIs* routines (e.g., glIsEnabled, glIsList, glIsTexture)

glGenLists	glDeleteLists	glFeedbackBuffer
glSelectBuffer	glRenderMode	glVertexPointer
glNormalPointer	glColorPointer	glIndexPointer
glReadPixels	glPixelStore	glGenTextures
glTexCoordPointer		glEdgeFlagPointer
glEnableClientState		glDisableClientState
glDeleteTextures		glAreTexturesResident
glFlush	glFinish	

An Example

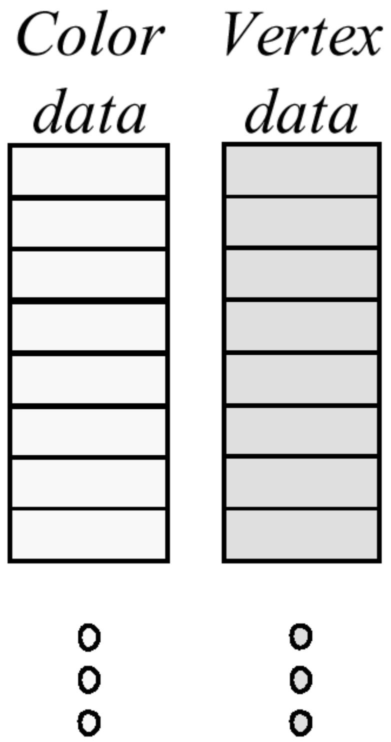
```
glNewList( CAR, GL_COMPILE );
  glCallList( CHASSIS );
  glTranslatef( … );
  glCallList( WHEEL );
  glTranslatef( … );
  glCallList( WHEEL );
      …
glEndList();
```



Vertex Arrays

- Pass arrays of vertices, colors, etc to OpenGL in a large chunk

```
glVertexPointer(3,GL_FLOAT,0,coords)
glColorPointer(4,GL_FLOAT,0,colors)
glEnableClientState(GL_VERTEX_ARRAY)
glEnableClientState(GL_COLOR_ARRAY)
glDrawArrays(GL_TRIANGLE_STRIP,0,numVerts);
```



- All active arrays are used in rendering
  - On: glEnalbleClientState()
  - Off: glDisableClientState()

Vertex Arrays

- Vertex Arrays allow vertices, and their attributes to be specified in chunks,
  - Not sending single vertices/attributes one call at a time.
- Three methods for rendering using vertex arrays:
  - glDrawArrays(): render specified primitive type by processing *nV* consecutive elements from enabled arrays.
  - glDrawElements(): indirect indexing of data elements in the enabled arrays. (shared data elements specified once in the arrays, but accessed numerous times)
  - glArrayElement(): processes a single set of data elements from all activated arrays. As compared to the two above, must appear between a glBegin()/glEnd() pair.

Vertex Arrays

- glDrawArrays(): draw a sequence
- glDrawElements(): methodically hop around
- glArrayElement(): randomly hop around

- glInterleavedArrays(): advanced call
  - can specify several vertex arrays at once.
  - also enables and disables the appropriate arrays

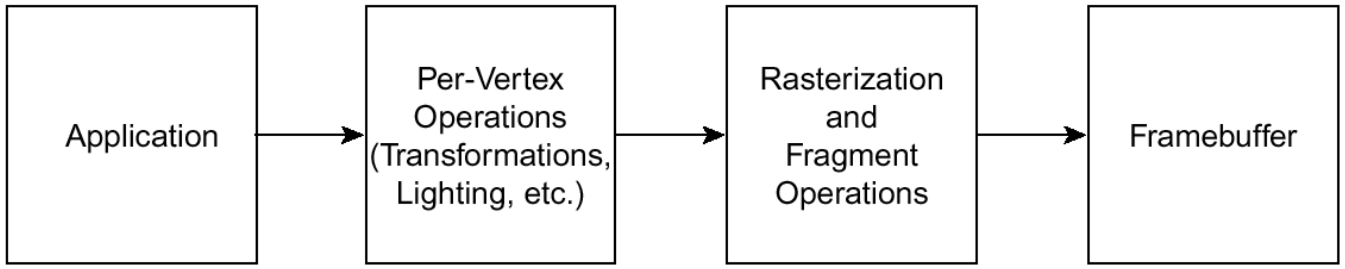
- Read Chapter 2 in Redbook for details of using vertex array

Why use Display lists or Vertex Arrays?

- May provide better performance than immediate mode rendering
  - Both are principally performance enhancements. On some systems, they may provide better performance than immediate mode because of reduced function call overhead or better data organization
    - format data for better memory access
  - Display lists can also be used to group similar sets of OpenGL commands, like multiple calls to glMaterial() to set up the parameters for a particular object
- Display lists can be shared between multiple OGL contexts
  - reduce memory usage or multi-context applications

Optimizing an OGL Application

- Which part of the OpenGL pipeline is performance bottleneck for your application


Application
Per-Vertex Operations (Transformations, Lighting, etc.)
Rasterization and Fragment Operations
Framebuffer

- Three possibilities:
  - Fill limited (check with reducing viewport size)
  - Geometry (transform) limited(check by replacing glVertex** calls to glNormal** calls)
  - Application limited (ogl commands don’t come fast enough, your data structure and data formats are at fault)

Reducing Per-pixel Operations (For fill-limited cases)

- Reduce the number of bits of resolution per color component.
  - E.g., reducing framebuffer depth from 24 bits to 15 bits for a 1280x1024 window, 37.5% reduction of the number of bits to fill (1.25 MBs)
- Reduce the number of pixels that need to be filled for geometric objects
  - Back face culling for convex shapes
- Utilize a lesser quality texture mapping minification filter
  - Use nearest filter
- Reduce the number of depth comparisons required for a pixel
  - ‘Hot spot’ analysis, use occlusion culling
- Utilize per-vertex fog, as compared to per-pixel fog

Reducing Per-Vertex Operations (For geometry-limited cases)

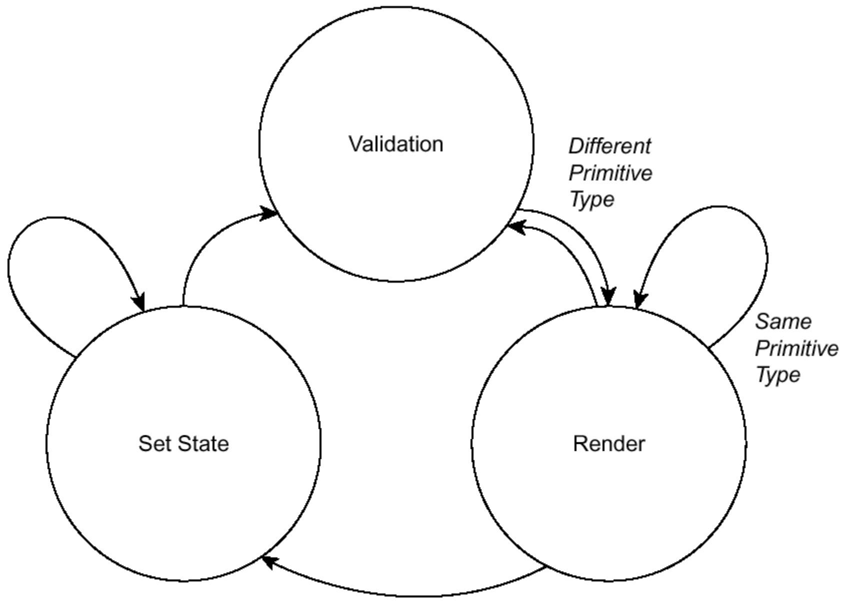
- The amount of computation done for a vertex can vary greatly depending upon which modes are enabled.
- Every vertex is
  - transformed
  - perspective divided
  - clip-tested
  - lighting
  - texture coordinate generation
  - user-defined clipping planes

Reducing Per-Vertex Operations (For geometry-limited cases)

- Determining the best way to pass geometry to the pipe
  - immediate mode, display list, vertex array, interleaved v-array?
- Use OpenGL transformation routines
  - ogl tracks the nature of top matrix on stack (don’t do full 4x4 if it’s just a 2D rotation)
  - So, use ogl transformation calls: glTranslate, glRotate, etc, instead of glMultiMatrix()
- Use connected primitives to save computation on OGL side
  - To avoid processing shared vertices repeatedly
- Or,… just don’t
  - Because minimizing number of validations that OGL has to do will save big time as well

Validation

- OpenGL is a state machine
- Validation is the operation that OpenGL utilizes to keep its internal state consistent with what the application has requested. Additionally, OpenGL uses the validation phase as an opportunity to update its internal caches, and function pointers to process rendering requests appropriately.
- For instance, glEnable requests a validation on the next rendering stage



Validation

- OGL ops that invoke validation

```
  glAccum()            glBegin()
glTexImage1D()  glTexSubImage1D()
glTexImage2D()  glTexSubImage2D()
glTexImage3D()  glTexSubImage3D()
glDrawPixels()    glCopyPixels()
glReadPixels()
```

- Object-oriented programming is a tremendous step in the quality of software engineering.
  - Unfortunately, OOP's "encapsulation" paradigm can cause significant performance problems if one chooses the obvious implementation for rendering geometric objects.

Example: say, we need to draw 10k squares in space

```
void drawCube( GLfloat color[], GLfloat* vertex[] )
{
  static int  idx[6][4] = {
    { ... },
  };

  glColor3fv( color );
  glBegin( GL_QUADS );
  for ( int i = 0; i < 6; ++i )
    for ( int j = 0; j < 4; ++j )
      glVertex3fv( vertex[idx[i][j]] );
  glEnd();
}
```

2.25 sec

```
void drawCube( GLfloat color[], GLfloat* vertex[]
{
  static int  idx[6][4] = {
    { ... },
  };

  glColor3fv( color );
  for ( int i = 0; i < 6; ++i )
    for ( int j = 0; j < 4; ++j )
      glVertex3fv( verts[idx[i][j]] );
}

glBegin( GL_QUADS );
for ( int i = 0; i < numCubes; ++i )
  drawCube( color[i], vertex[8*i] );
glEnd();
```

1.00 sec

```
void drawCube( GLfloat color[], GLfloat* vertex[] )
{
  static int  idx[6][4] = {
    { ... },
  };

  glColor3fv( color );
  glBegin( GL_QUADS );
  for ( int i = 0; i < 4; ++i )   // Render top of cube
    glVertex3fv( vertex[idx[0][i]] );
  for ( i = 0; i < 4; ++i )  // Render bottom of cube
    glVertex3fv( vertex[idx[1][i]] );
  glEnd();

  glBegin( GL_QUAD_STRIP );
  for ( i = 2; i < 6; ++i ) {
    glVertex3fv( vertex[idx[i][0]] );
    glVertex3fv( vertex[idx[i][1]] );
  }
  glVertex3fv( vertex[idx[2][0]] );
  glVertex3fv( vertex[idx[2][1]] );
  glEnd();
}
```

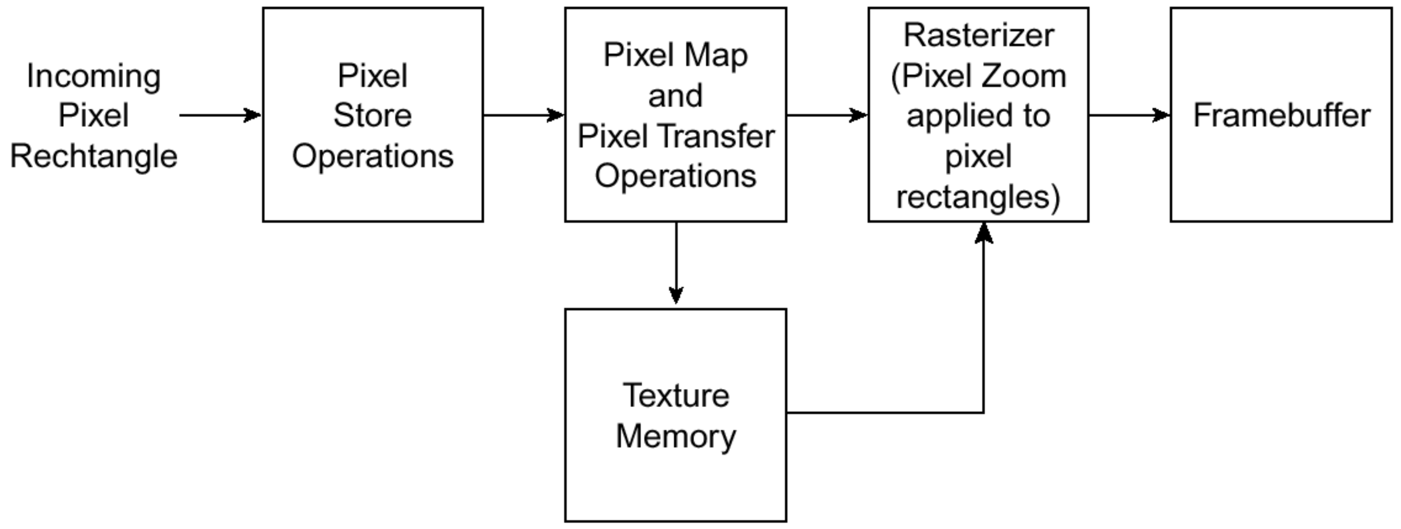
2.13 sec

General Techniques

- State sorting
  - Sort the render requests and state settings based upon the penalty for setting that particular part of the OpenGL state.
  - For example, loading a new texture map is most likely a considerably more intensive task than setting the diffuse material color, so attempt to group objects based on which texture maps they use, and then make the other state modifications to complete the rendering of the objects.

General Techniques (2)

- When sending pixel type data down to the OpenGL pipeline, try to use pixel formats that closely match the format of the framebuffer, or requested internal texture format.
  - Conversion takes time


Incoming Pixel Rechtangle
Pixel Store Operations
Pixel Map and Pixel Transfer Operations
Rasterizer (Pixel Zoom applied to pixel rectangles)
Framebuffer
Texture Memory

General Techniques (3)

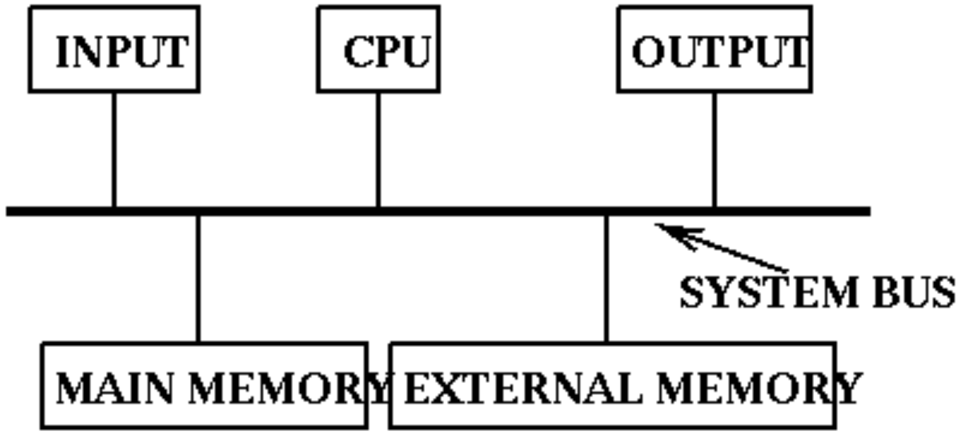
- Pre-transform static objects
  - For objects that are permanently positioned in world coordinates pre-transforming the coordinates of such objects as compared to calling glTranslate*() or other modeling transforms can represent a saving.

General Techniques (4)

- Use texture objects (ogl v1.1)
- Use texture proxies to verify that a given texture map will fit into texture memory
- Reload textures using `glTexSubImage*D()`
  - Calls to glTexImage*D() request for a texture to be allocated, and if there is a current texture present, deallocate it

Graphics Architecture

- Computer Architecture (the theoretical one)



PC Architecture - Buses

- The Processor Bus: highest-level bus that the chipset uses to send information to and from the processor.
- The Cache Bus: a dedicated bus for accessing the system cache. Aka backside bus.
- The Memory Bus: a system bus that connects the memory subsystem to the chipset and the processor. In some systems, processor bus and memory bus are basically the same thing
- The Local I/O Bus: a high-speed input/output bus (closer or even on the memory bus directly, so local to proc) used for connecting performance-critical peripherals (video card/high speed disks/high speed NIC) to the memory, chipset, and processor. (e.g. PCI, VESA)
- The Standard I/O Bus: used for slower peripherals (mice, modems, regular sound cards, low-speed networking) and also for compatibility with older devices, say, ISA, EISA
- Another classification: internal/external (expansion) bus

Some Buses

ISA	Industry Standard Architecture	8, 16	1980
MCA	Micro Channel Architecture	16, 32	1987
EISA	Extended ISA	32	1988
VESA	Video Electronics Standard Association	32	1992
PDS	Processor Direct Slot (Macintosh)	32	1993
PCI	Peripheral Component Interconnect	32, 64	1993
PCMCIA	Personal Computer Memory Card International Association	8,16,32	1992

System Chipset

- The system chipset and controllers are the logic circuits that are the intelligence of the motherboard
- A chipset is just a set of chips.

- At one time, most of the functions of the chipset were performed by multiple, smaller controller chips. There was a separate chip (often more than one) for each function: controlling the cache, performing DMA, handling interrupts, transferring data over the I/O bus, etc.

- Over time these chips were integrated to form a single set of chips, or chipset, that implements the various control features on the motherboard

A New Addition to the Bus Family -AGP

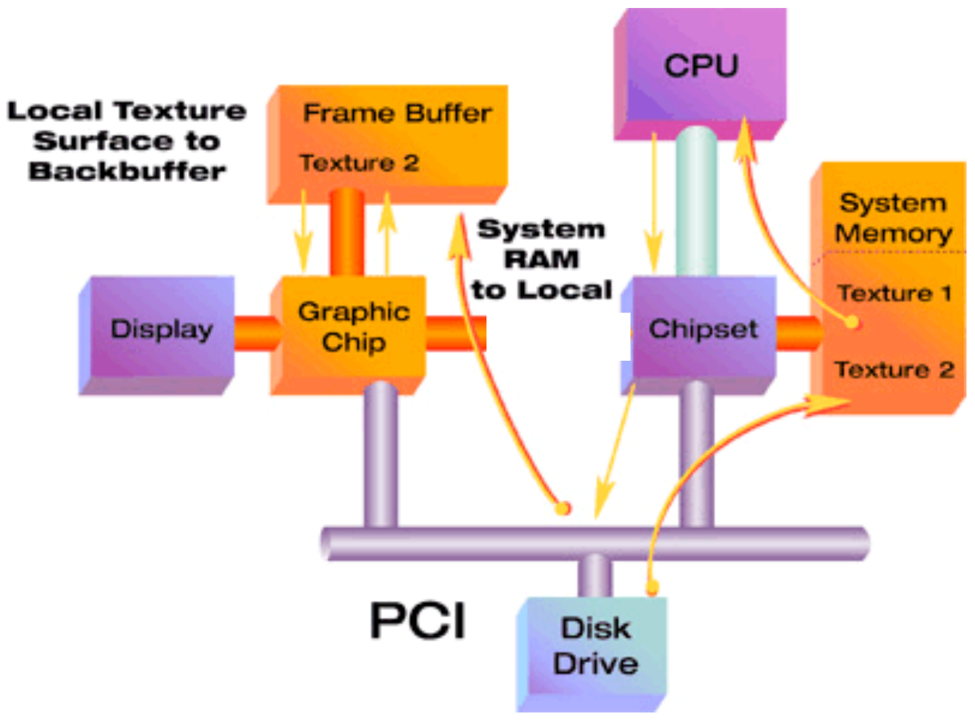
- Advanced Graphic Port devised in 1997 by Intel
- AGP: 32-bit Bus designed for the high demands of 3-D graphics, based on the PCI 2.1 standard.
  - deliver a peak bandwidth higher than the PCI bus using pipelining, sideband addressing, and more data transfers per clock.
  - also enables graphics cards to execute texture maps directly from system memory instead of forcing it to pre-load the texture data to the graphics card's local memory.

Bus Specs

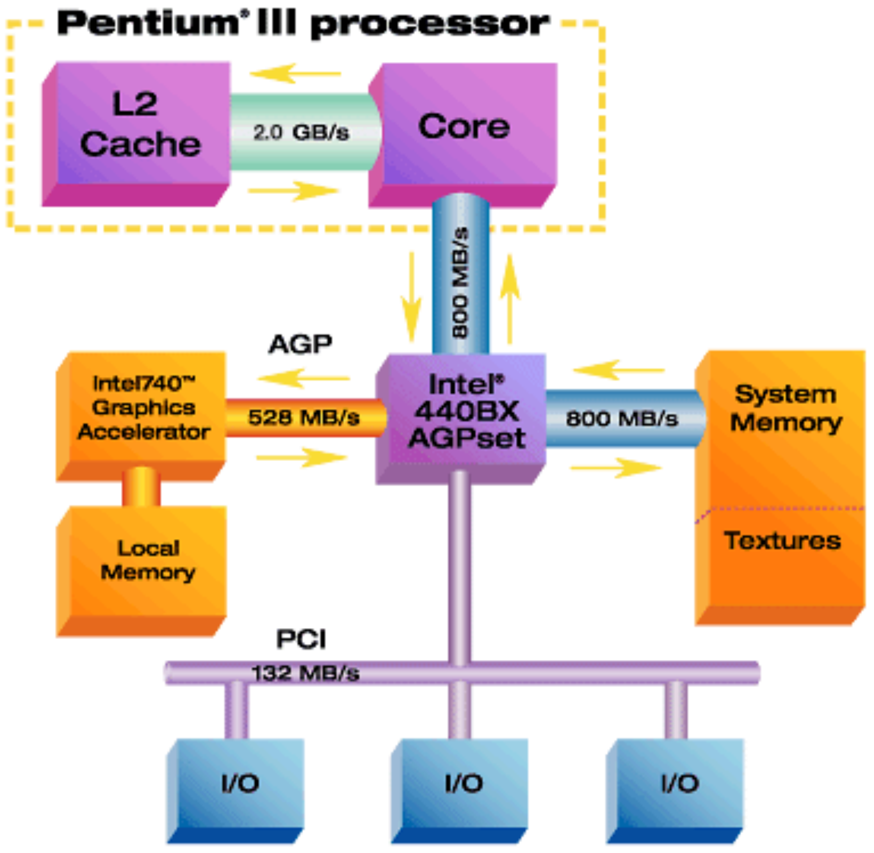
BUS	Bits	Clock	Bandwidth (MB/s)
8-bit ISA	8	8.3	7.9
16-bit ISA	16	8.3	15.9
EISA	32	8.3	31.8
VLB	32	33	127.2
PCI	32	33	127.2
64-bit PCI 2.1	64	66	508.6
AGP	32	66	254.3
AGP (x2 mode)	32	66x2	508.6
AGP (x4 mode)	32	66x4	1,017.3

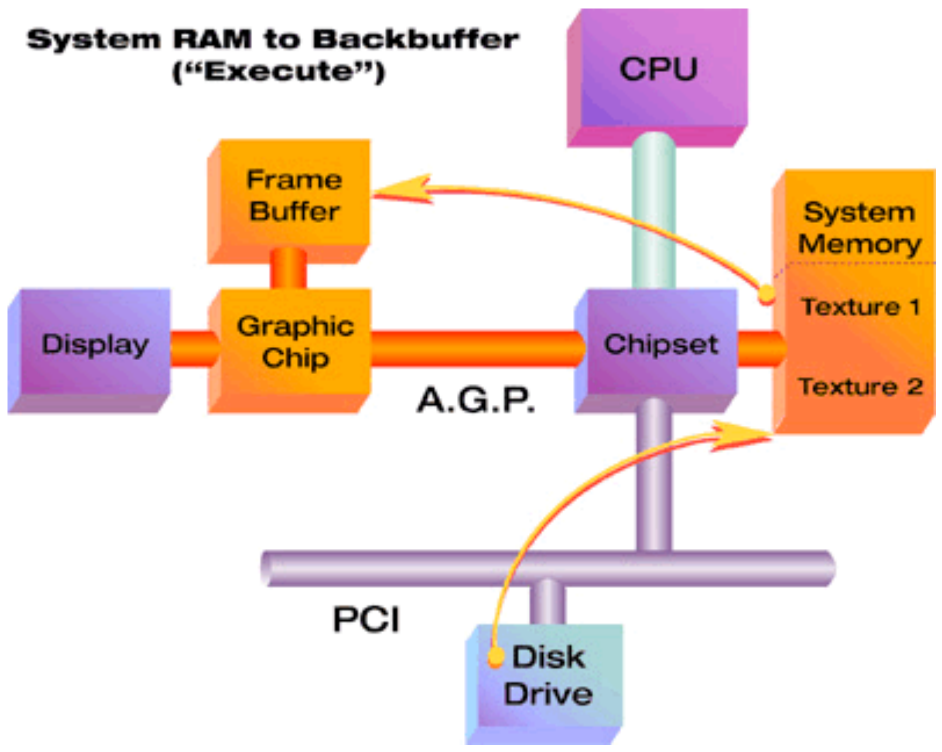
PCIe v3.0 x16 (lanes): 8x16 bits, 4GHz, 16GB/s

Pre-AGP times, say we want to do a texture mapping



With AGP, in a PIII system





Rules of Thumb

- The underlying architecture have impacts on the application development
- New applications drive the evolution of architecture