An Interactive Introduction to OpenGL Programming

Dr. Mohammad Iqbal

Based-on slide : Dave Shreiner, Ed Angel, Vicki Shreiner

Agenda

- General OpenGL Introduction
- Rendering Primitives
- Rendering Modes
- Lighting
- Texture Mapping
- Additional Rendering Attributes
- Imaging

Goals for Today

- Tujuan : mendemonstrasikan OpenGL dalam menghasilkan program grafik interaktif dalam
  - Membuat model 3D obyek atau imagery
  - Lighting - pencahayaan
  - texture mapping
- Mengenalkan topik lanjut untuk pendalaman berikutnya

OpenGL and GLUT Overview

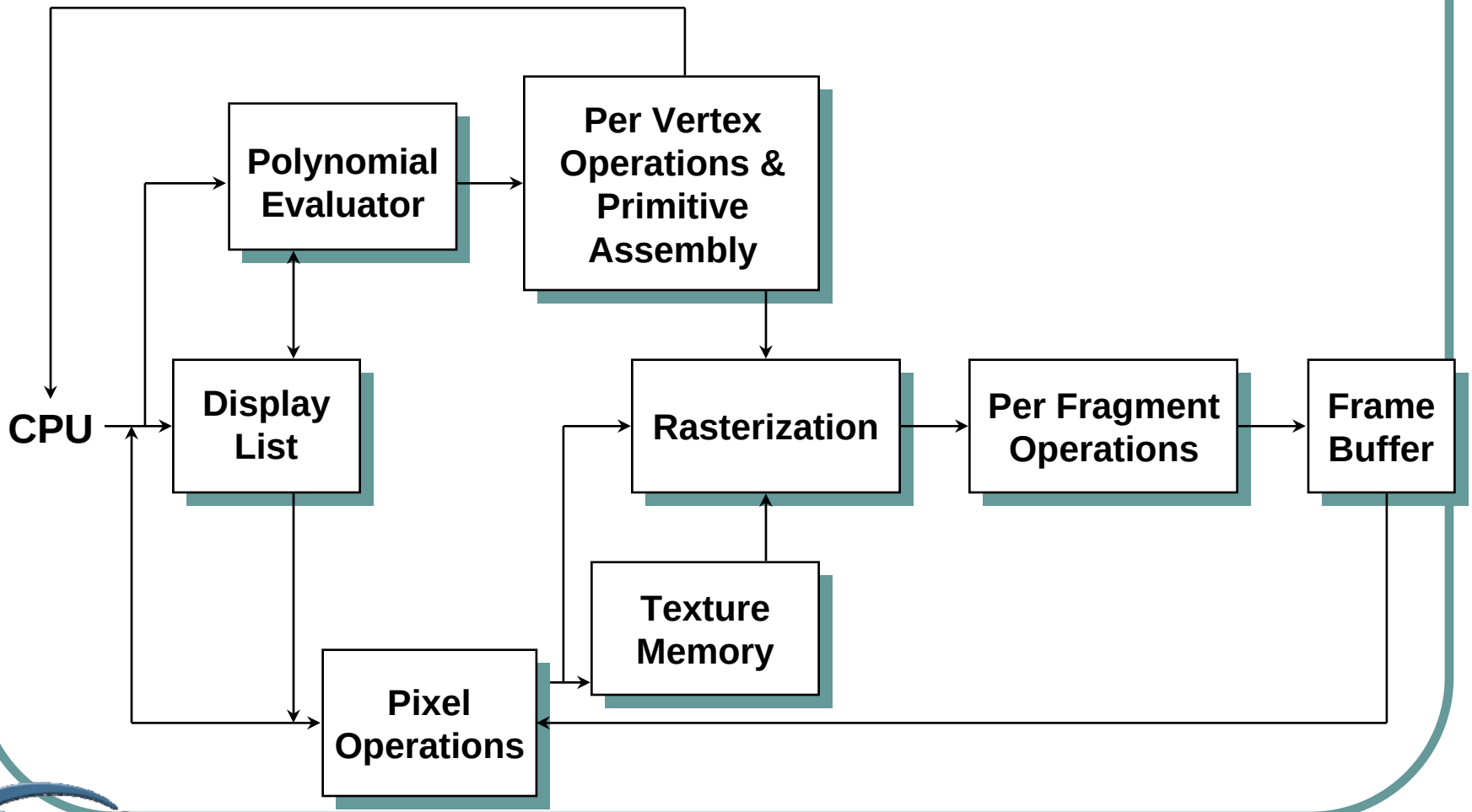
OpenGL and GLUT Overview

- Apakah OpenGL & apa manfaatnya?
- OpenGL dalam sistem window
- Mengapa GLUT
- Template program GLUT

Apakah OpenGL?

- Graphics rendering API (Application Programming Interface)
  - citra warna *high-quality* yang terdiri dari geometric dan citra primitif
  - Independen window system
  - Independen operating system

Arsitektur OpenGL



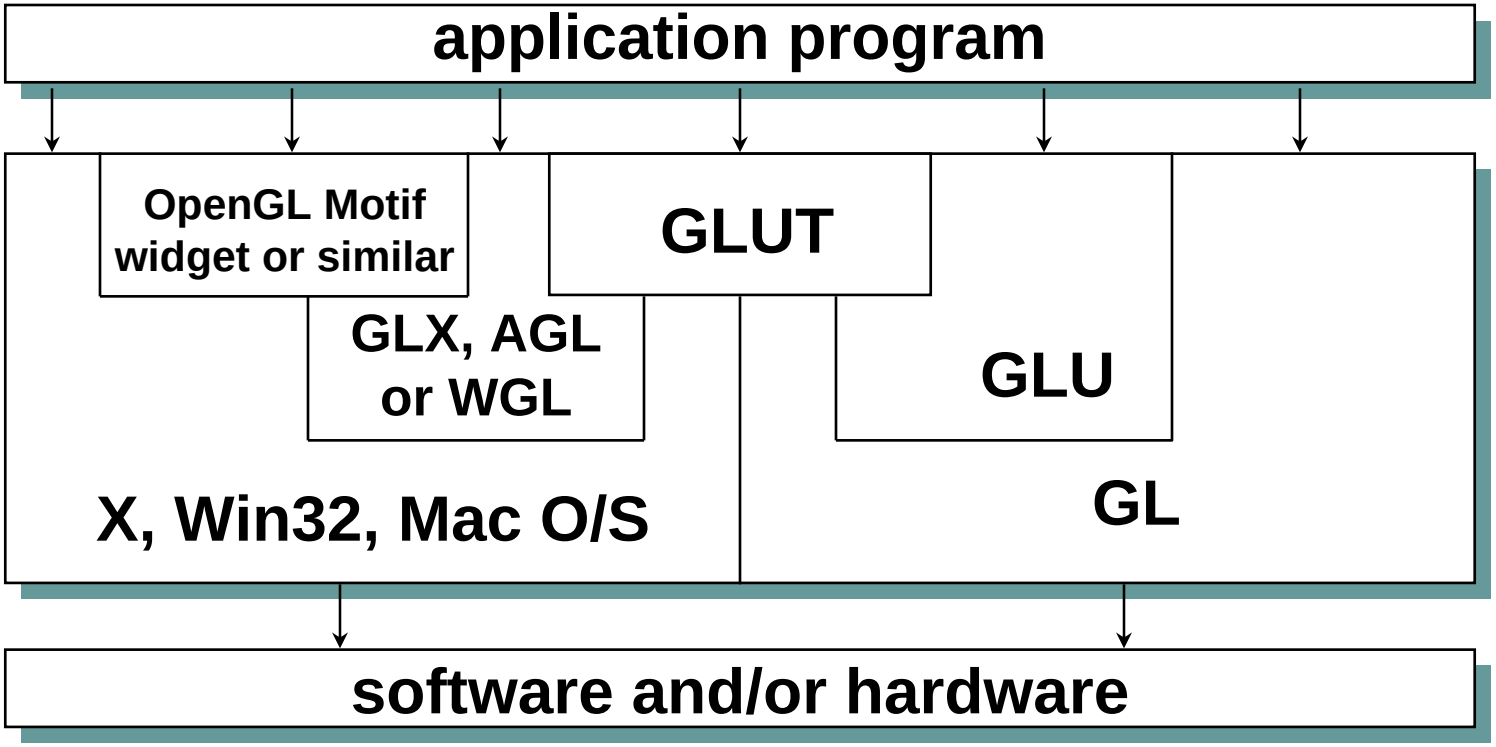
OpenGL sebagai Renderer

- Geometric primitif
  - titik, garis dan poligon
- Image Primitif
  - Citra dan bitmap
  - Memisahkan pipeline untuk citra dan geometry
    - linked ketika penerapan texture mapping
- Rendering tergantung pada status
  - warna, material, light sources, dll.

API yang Terkait OpenGL

- AGL, GLX, WGL
  - Perekat antara OpenGL dan sistem window
- GLU (OpenGL Utility Library)
  - Bagian OpenGL
  - NURBS, tessellators, quadric shapes, dll.
- GLUT (OpenGL Utility Toolkit)
  - portable windowing API
  - Bukan bagian OpenGL secara ofisial

API yang Terkait OpenGL



Preliminaries

- Headers Files
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GL/glut.h>
- Libraries
- Enumerated Types
  - OpenGL defines numerous types for compatibility
    - GLfloat, GLint, GLenum, etc.

Dasar-dasar GLUT

- Struktur Aplikasi
  - Configure dan open window
  - Initialisasi status OpenGL
  - Register fungsi input callback
    - render
    - resize
    - input: keyboard, mouse, dll.
  - Enter event processing loop

Contoh Program

```
void main( int argc, char** argv )
{
  int mode = GLUT_RGB|GLUT_DOUBLE;
  glutInitDisplayMode( mode );
  glutCreateWindow( argv[0] );
  init();
  glutDisplayFunc( display );
  glutReshapeFunc( resize );
  glutKeyboardFunc( key );
  glutIdleFunc( idle );
  glutMainLoop();
}
```

Inisialisasi OpenGL

- Set up status yg ingin digunakan

```
void init( void )
{
  glClearColor( 0.0, 0.0, 0.0, 1.0 );
  glClearDepth( 1.0 );

  glEnable( GL_LIGHT0 );
  glEnable( GL_LIGHTING );
  glEnable( GL_DEPTH_TEST );
}
```

Fungsi input Callback GLUT

- Rutin yang dipanggil ketika sesuatu terjadi
  - window resize atau redraw
  - user input
  - animasi
- Me-“register” callbacks pada GLUT

```
glutDisplayFunc( display );
glutIdleFunc( idle );
glutKeyboardFunc( keyboard );
```

Rendering Callback

- Lakukan penggambaran grafik pada bagian ini :

```
glutDisplayFunc( display );

void display( void )
{
  glClear( GL_COLOR_BUFFER_BIT );
  glBegin( GL_TRIANGLE_STRIP );
    glVertex3fv( v[0] );
    glVertex3fv( v[1] );
    glVertex3fv( v[2] );
    glVertex3fv( v[3] );
  glEnd();
  glutSwapBuffers();
}
```

Idle Callbacks

- Gunakan untuk animasi dan update yang continyu

```
glutIdleFunc( idle );
```

```
void idle( void )
{
  t += dt;
  glutPostRedisplay();
}
```

User Input Callbacks

- Untuk pemrosesan input dari user

```
glutKeyboardFunc( keyboard );
```

```
void keyboard( unsigned char key, int x, int y )
{
  switch( key ) {
    case ‘q’ : case ‘Q’ :
      exit( EXIT_SUCCESS );
      break;
    case ‘r’ : case ‘R’ :
      rotate = GL_TRUE;
      glutPostRedisplay();
      break;
  }
}
```

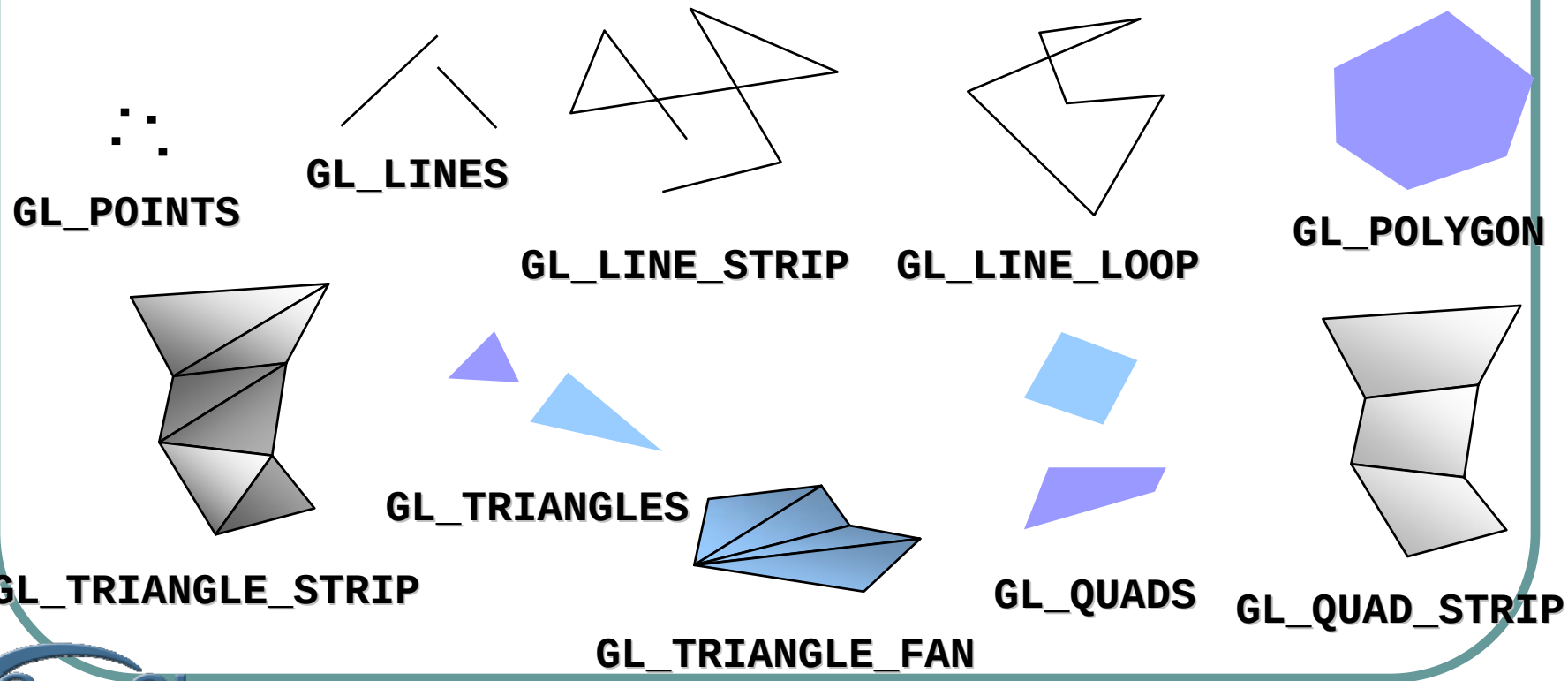
Elementary Rendering

Agenda Elementary Rendering

- Geometric Primitives
- Managing OpenGL State
- OpenGL Buffers

OpenGL Geometric Primitif

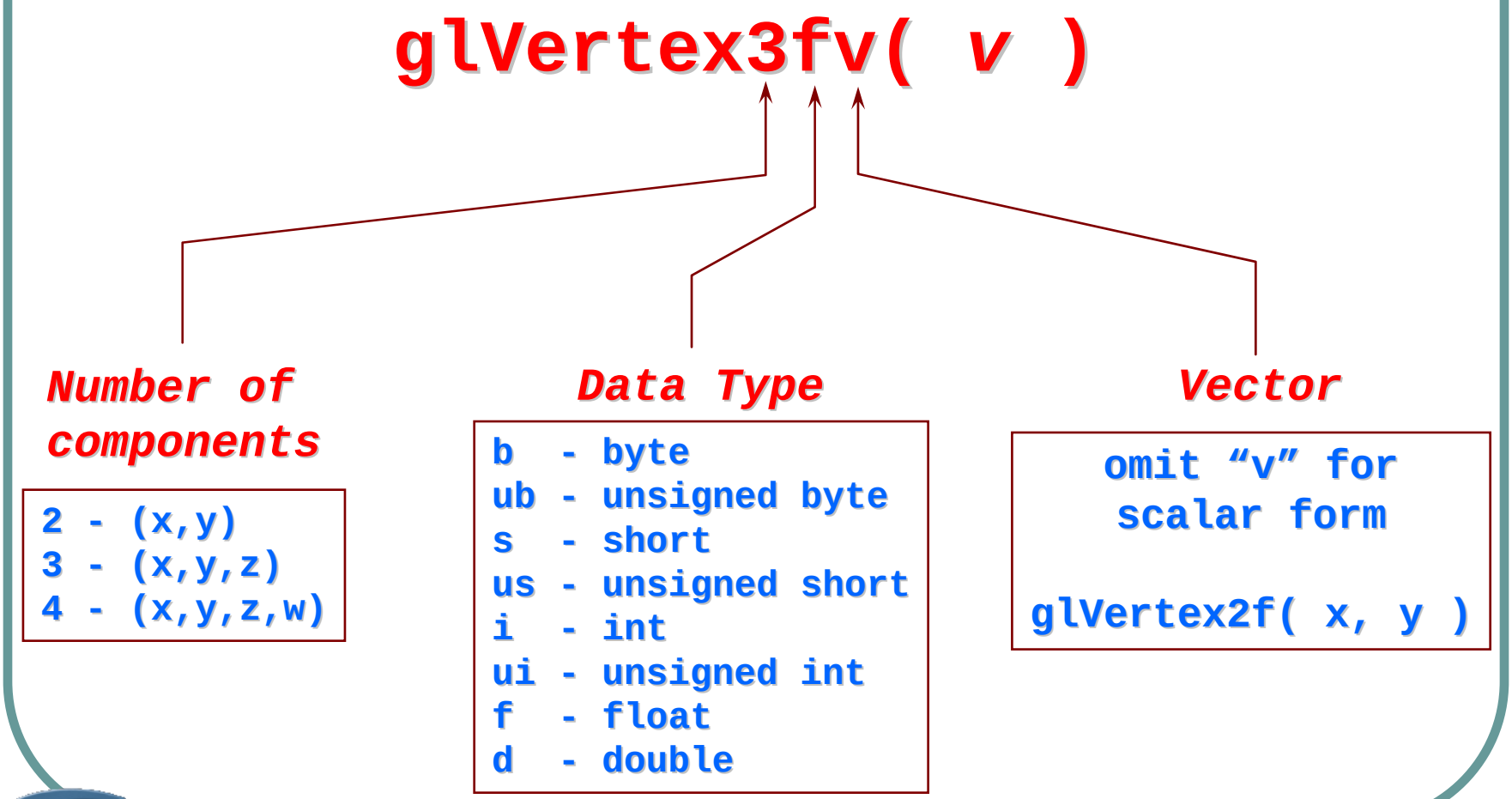
- Semua jenis geometric primitif dispesifikasikan oleh pipeline vertices



Contoh Sederhana

```
void drawRhombus( GLfloat color[] )
{
    glBegin( GL_QUADS );
    glColor3fv( color );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 1.5, 1.118 );
    glVertex2f( 0.5, 1.118 );
    glEnd();
}
```

Format Perintah OpenGL



Specifying Geometric Primitives

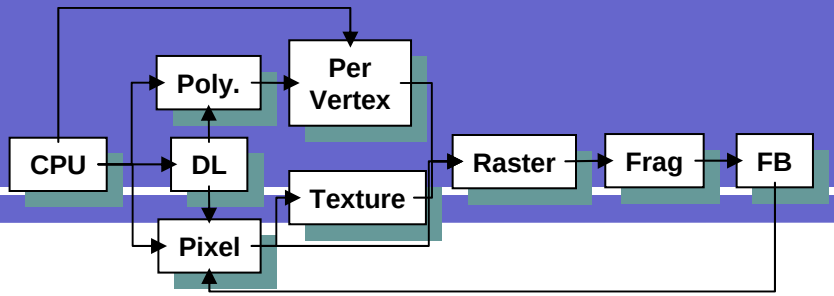
- Primitif dispesifikasikan menggunakan :

```
glBegin( primType );
glEnd();
```

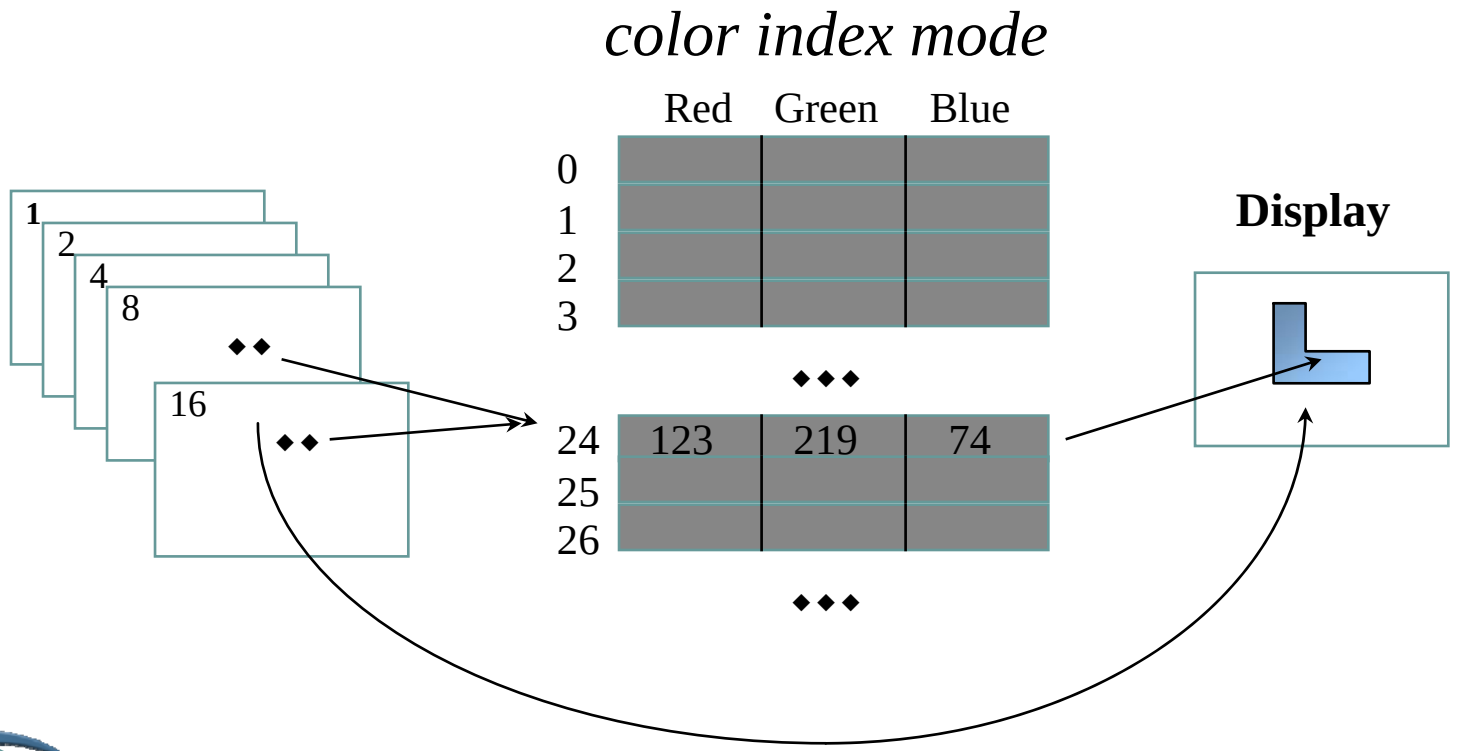
  - *primType* menentukan bagaimana *vertice* dikombinasikan

```
GLfloat red, green, blue;
Glfloat coords[3];
glBegin( primType );
for ( i = 0; i < nVerts; ++i ) {
  glColor3f( red, green, blue );
  glVertex3fv( coords );
}
glEnd();
```

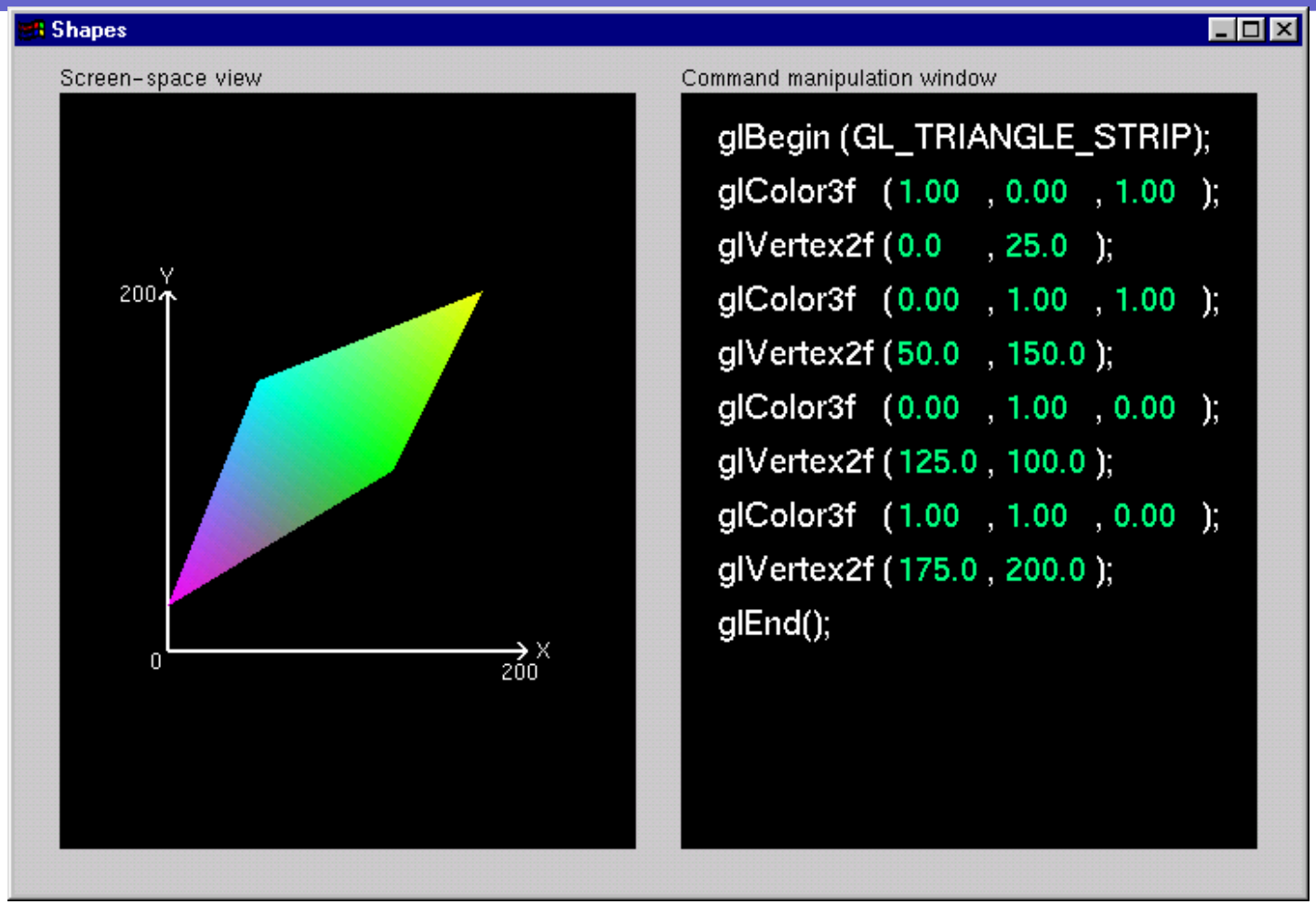
Model Warna OpenGL



- RGBA atau Color Index

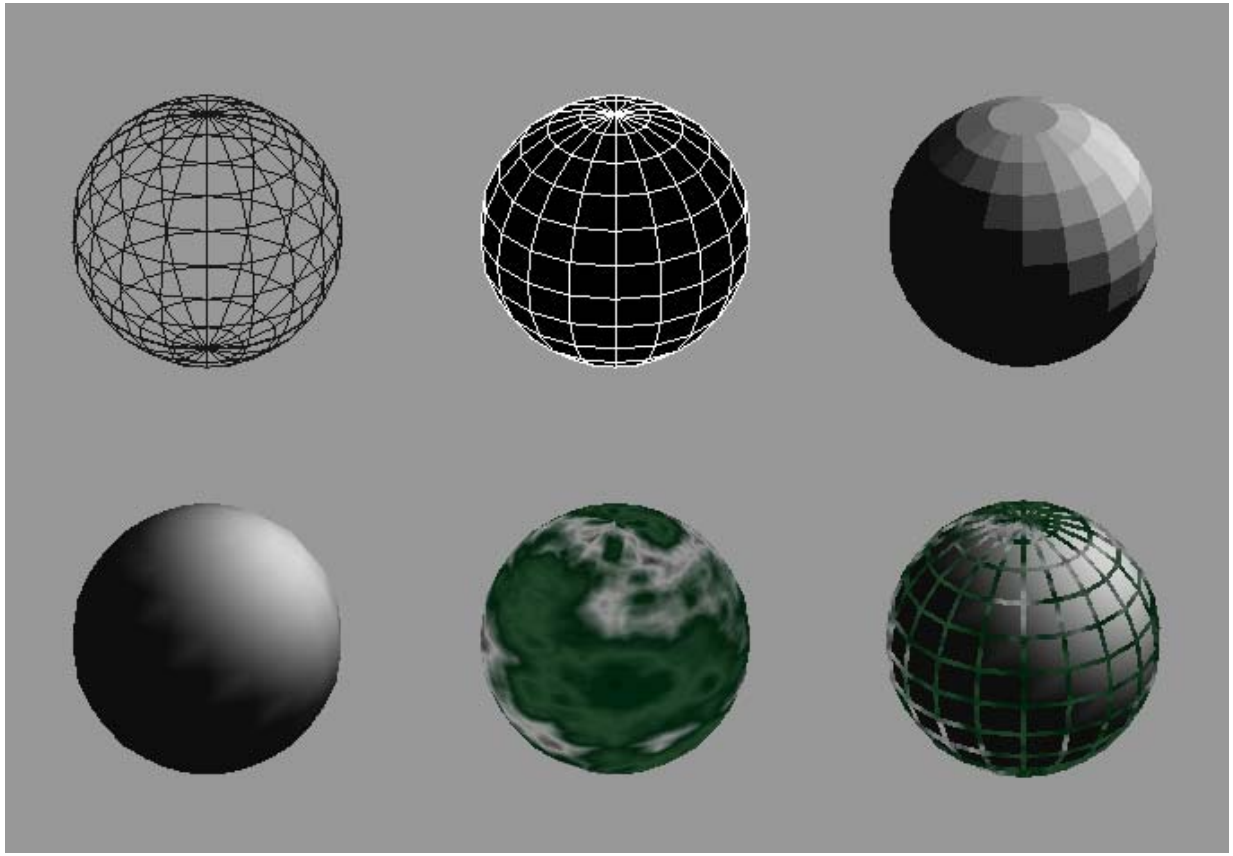


Shapes Tutorial



Mengendalikan tampilan *Appearance* (Rendering)

Dari Wireframe menjadi Texture Mapped



Mesin Status OpenGL

- Setiap atribut rendering di encapsulapsi dalam *OpenGL State*
  - rendering styles
  - shading
  - lighting
  - texture mapping

Manipulasi Status OpenGL

- Tampilan dikendalikan oleh status terakhir

```
for each ( primitive to render ) {
    update OpenGL state
    render primitive
}
```

- Manipulasi atribut vertex adalah cara umum untuk memanipulasi status

```
glColor*() / glIndex*()
glNormal*()
glTexCoord*()
```

Mengendalikan Status terakhir

- Setting Status

```
glPointSize( size );
glLineStipple( repeat, pattern );
glShadeModel( GL_SMOOTH );
```

- Enabling Features

```
glEnable( GL_LIGHTING );
glDisable( GL_TEXTURE_2D );
```

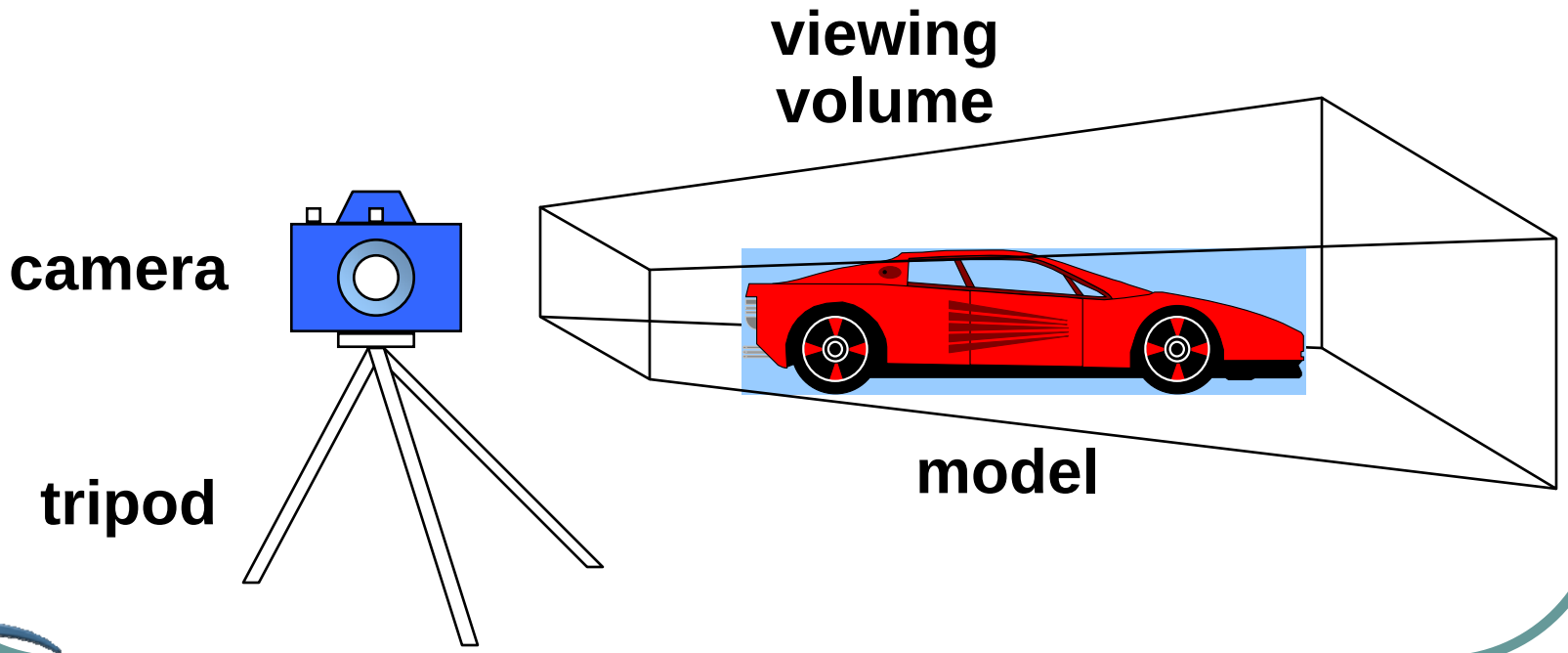
Transformasi

Transformasi dalam OpenGL

- Modeling
- Viewing
  - orientasi kamera
  - projection
- Animasi
- Map to screen

Analogi Kamera

- 3D adalah seperti mengambil citra pada fotografi (Banyak foto!)



Analogi Kamera dan Transformasi

- Projection transformations
  - Mengatur lensa kamera
- Viewing transformations
  - Mengatur posisi *tripod* dan orientasi viewing suatu volume dalam dunia nyata (*world*)
- Modeling transformations
  - memindahkan model
- Viewport transformations
  - Memperbesar atau mengurangi fotografi secara fisik

Sistem Koordinat dan Transformasi

- Langkah dalam menyiapkan citra
  - spesifikasikan geometri (world coordinates)
  - spesifikasikan kamera (camera coordinates)
  - proyeksi (window coordinates)
  - Petakan ke viewport (screen coordinates)
- Setiap langkah menggunakan transformasi
- Setiap transformasi adalah *ekuivalen* pada perubahan di sistem koordinat (frames)

Transformasi Affine

- Transformasi yang mempertahankan bentuk geometri
  - garis, poligon, quadric
- Affine = mempertahankan garis (line preserving)
  - Rotasi, translasi, skala
  - Proyeksi
  - Konkatnasi (komposisi)

Koordinat Homogen

- Setiap vertex adalah vector kolom

$$\vec{v} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- w umumnya bernilai 1.0
- Semua operasi adalah perkalian matriks
- Arah (directed line segments) direpresentasikan w = 0.0

3D Transformations

- Vertex ditransformasikan oleh 4 x 4 matrik
  - Semua operasi affine adalah perkalian matriks
  - Semua matrik disimpan secara *column-major* dalam OpenGL
  - matriks selalu dalam kondisi *post-multiplied*
  - Produk matrik dan vector adalah $\mathbf{M}\vec{v}$

$$\mathbf{M} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

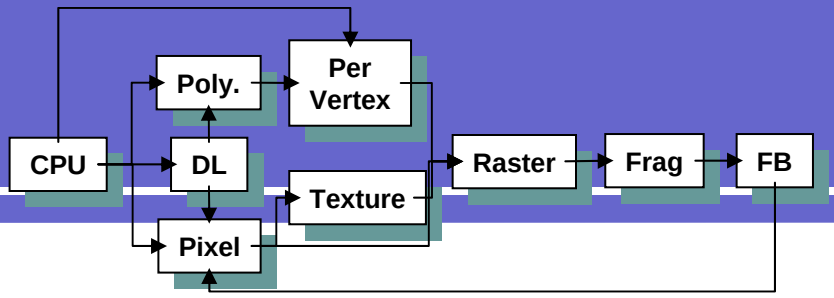
Menspesifikasikan Transformasi

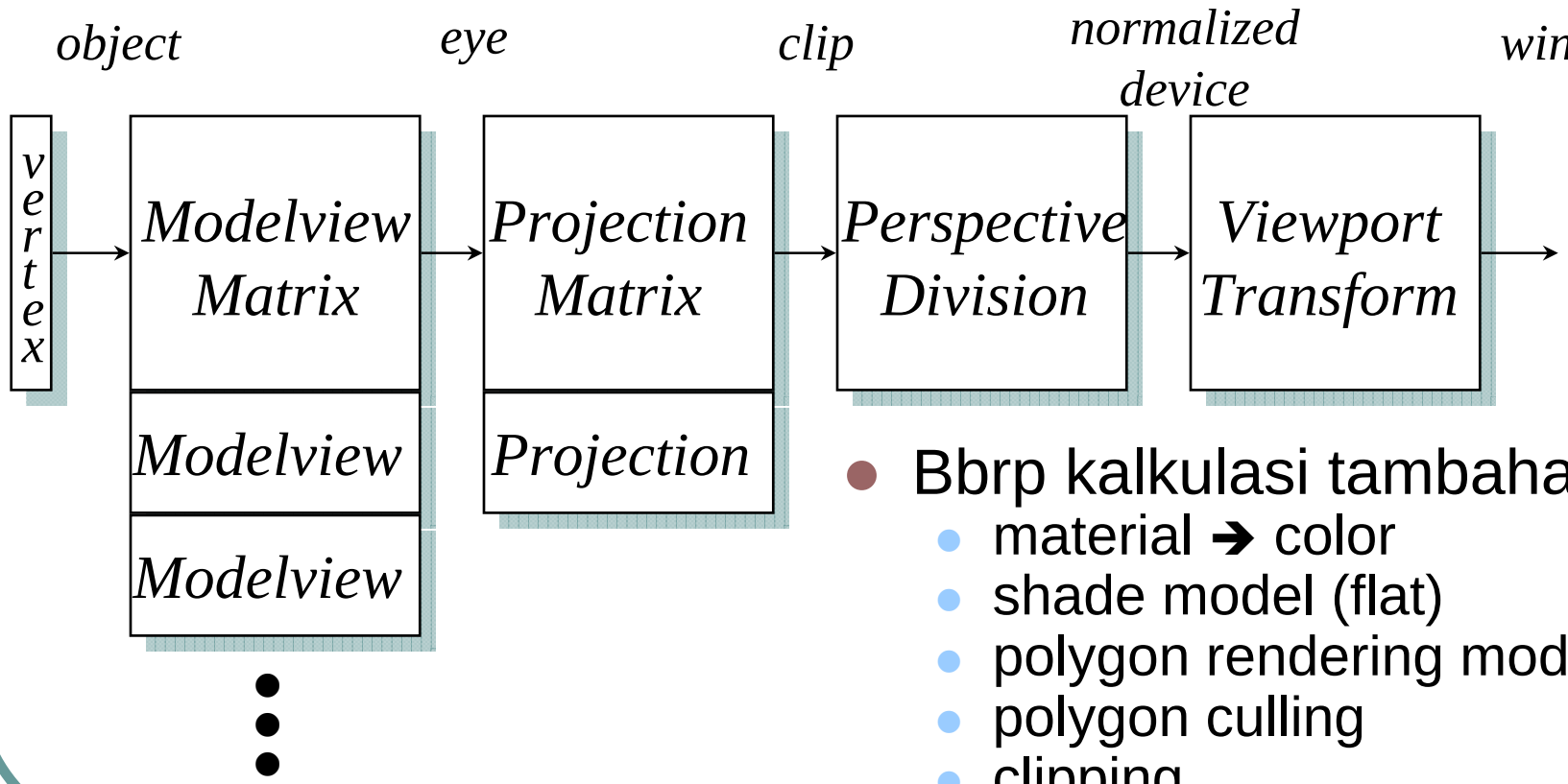
- Programmer memiliki 2 styles untuk men-spesifikasikan transformasi
  - spesifikasikan matriks (**`glLoadMatrix, glMultMatrix`**)
  - spesifikasikan operasi (**`glRotate, glOrtho`**)
- Programmer tidak perlu mengingat jenis matriksnya secara tepat
  - cek lampiran buku Red Book (Programming Guide)

Programming Transformations

- Sebelum proses rendering, view, locate, dan orientasi :
  - Posisi mata/kamera
  - 3D geometri
- Mengatur matriks
  - Termasuk matriks stack
- Kombinasikan (composite) transformasi

Pipeline Transformasi





- Bbrp kalkulasi tambahan :
  - material ➔ color
  - shade model (flat)
  - polygon rendering mode
  - polygon culling
  - clipping

Operasi Matriks

- Spesifikasikan Matriks Stack terkini

  `glMatrixMode( GL_MODELVIEW atau GL_PROJECTION )`

- Matriks atau operasi Stack lain

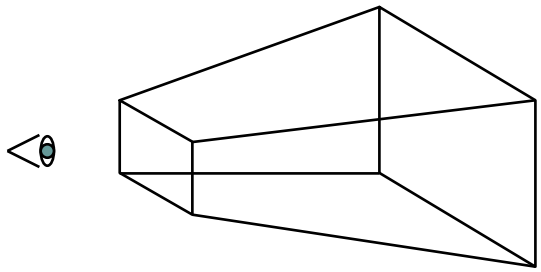
  `glLoadIdentity()` `glPushMatrix()`
  `glPopMatrix()`

- Viewport
  - Biasanya sama dengan ukuran window size
  - aspek rasio viewport harus sama dengan transformasi proyeksi atau citra hasilnya akan terdistorsi

    `glViewport( x, y, width, height )`

Transformasi Proyeksi

Bentuk untuk menampilkan (viewing) frustum :



- Perspective projection

  `gluPerspective( fovy, aspect, zNear, zFar )`

  `glFrustum( left, right, bottom, top, zNear, zFar )`

- Orthographic parallel projection

  `glOrtho( left, right, bottom, top, zNear, zFar )`

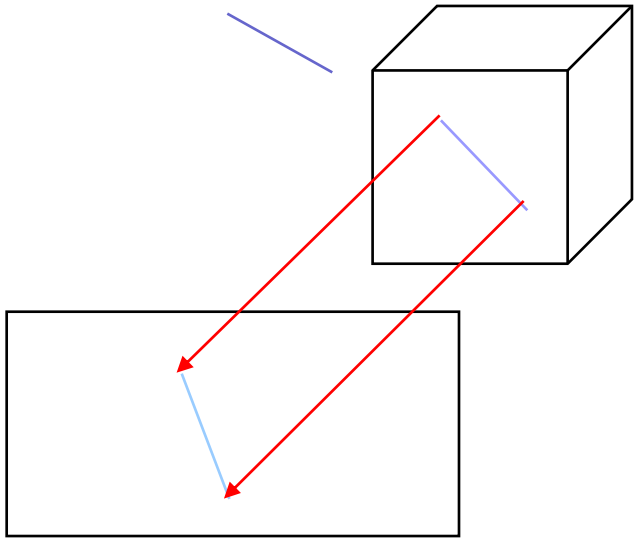
  `gluOrtho2D( left, right, bottom, top )`

  - Gunakan **glOrtho** dengan nilai z mendekati nol

Mengaplikasikan Transformasi Proyeksi

- Pengunaan Umum (orthographic projection)

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glOrtho( left, right, bottom, top, zNear, zFar
  );
```

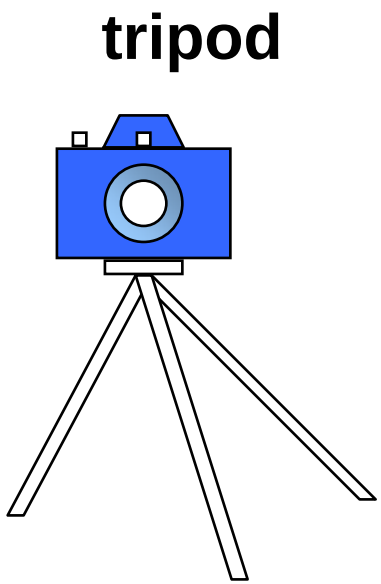


Viewing Transformations

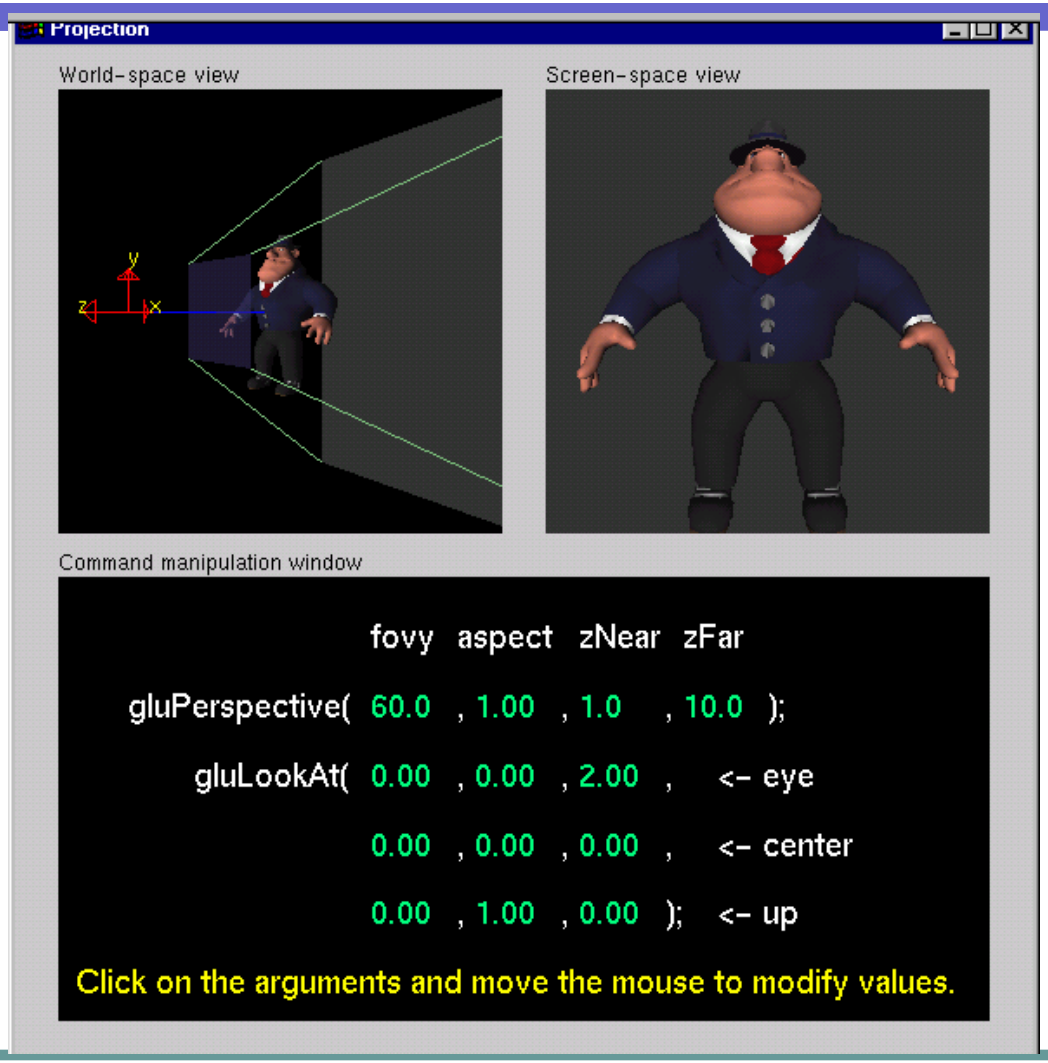
- Posisi Kamera/mata dalam *scene*
  - Posisikan tripod (eye) ; persiapkan (aim) Kamera
- Untuk Scene “fly through”
  - Ubah transformasi viewing dan *re-draw* scene

```
gluLookAt( eyex, eyey, eyez,
           aimx, aimy, aimz,
           upx, upy, upz )
```

  - *up vector* menghasilkan orientasi unik
  - Berhati-hati dalam *de-generate* posisi



Projection Tutorial



Modeling Transformations

- Memindahkan obyek

  `glTranslate{fd}( x, y, z )`

- Rotasi obyek di sekitar sumbu utama $(x \quad y \quad z)$

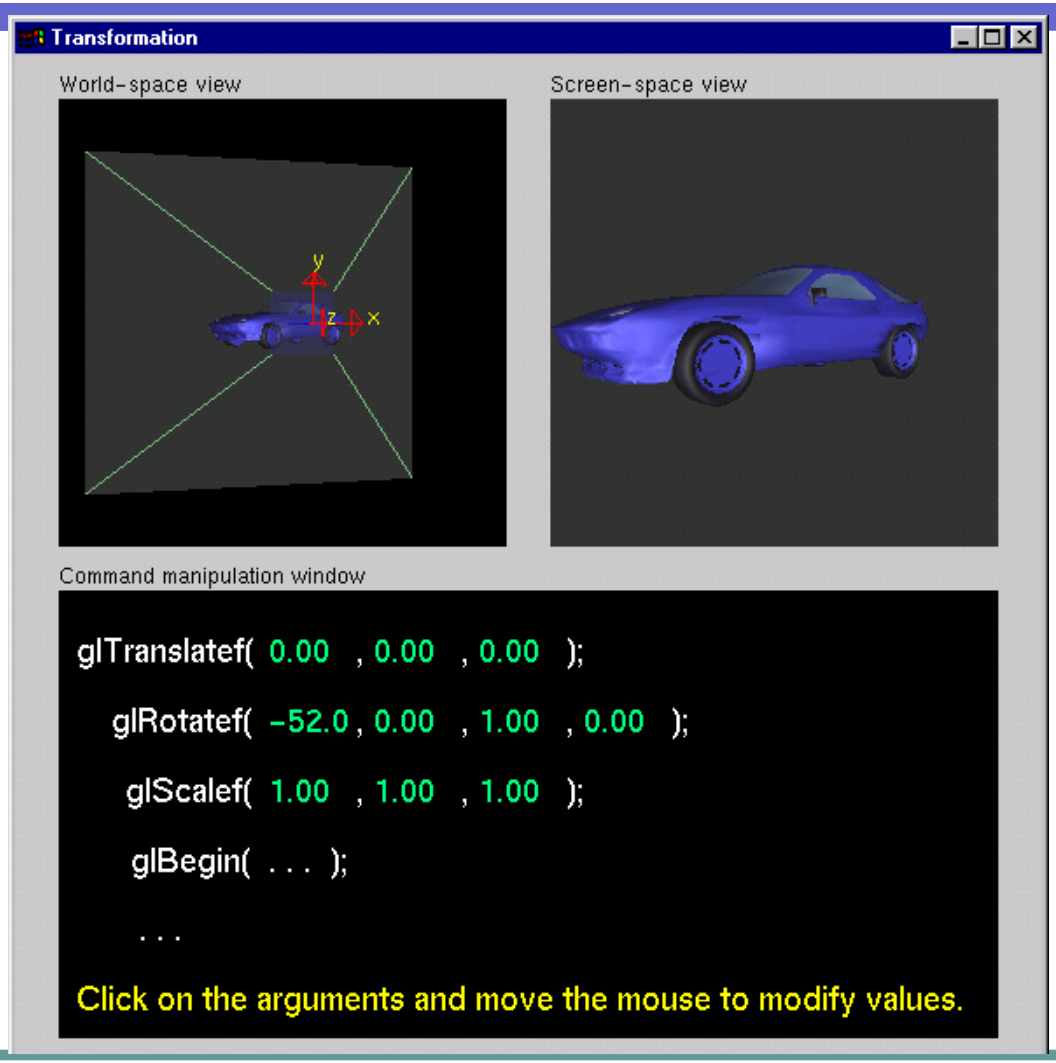
  `glRotate{fd}( angle, x, y, z )`

  - angle dalam derajat

- Dilasi (stretch atau shrink) atau mirror object

  `glScale{fd}( x, y, z )`

Transformation Tutorial



Connection: Viewing dan Modeling

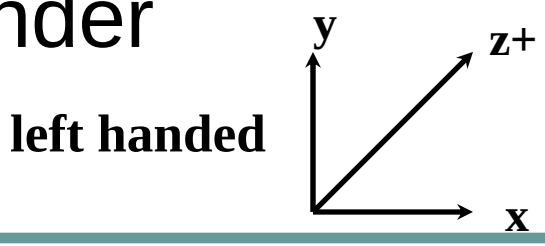
- Memindahkan kamera equivalent dengan memindahkan setiap obyek di dunia nyata di depan kamera diam
- Viewing transformation equivalent dengan transformasi modeling

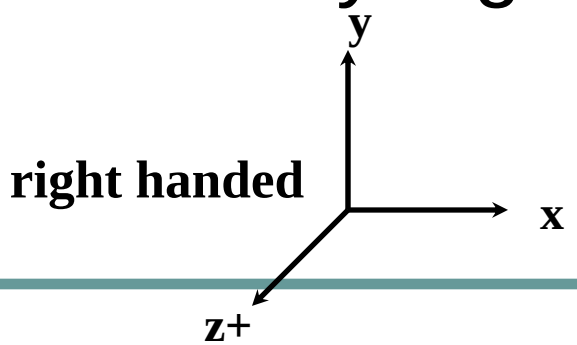
  `gluLookAt()` memiliki perintah tersendiri

  yaitu *polar view* atau *pilot view*

Proyeksi dengan kaidah tangan kiri

- Projection transformations (`gluPerspective, glOrtho`) berdasarkan kaidah tangan kiri
  - bayangkan ***`zNear`*** dan ***`zFar`*** sebagai jarak tertentu dari view point
- Setiap hal selain itu adalah kaidah tangan kanan, termasuk vertex yang di render





Penggunaan Umum Transformasi

- 3 contoh rutin resize()
  - Re-status proyeksi & viewing transformations
- Umumnya dipanggil ketika window resize
- Di-register sebagai callback untuk `glutReshapeFunc()`

resize(): Perspective & LookAt

```
void resize( int w, int h )
{
   glViewport( 0, 0, (GLsizei) w, (GLsizei) h
  );
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   gluPerspective( 65.0, (GLdouble) w / h,
                   1.0, 100.0 );
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity();
   gluLookAt( 0.0, 0.0, 5.0,
              0.0, 0.0, 0.0,
              0.0, 1.0, 0.0 );
}
```

resize(): Perspective & Translate

Efek yang sama dengan LookAt :

```
void resize( int w, int h )
{
   glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   gluPerspective( 65.0, (GLdouble) w/h,
                   1.0, 100.0 );
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity();
   glTranslatef( 0.0, 0.0, -5.0 );
}
```

resize( ): Ortho (bagian 1)

```
void resize( int width, int height )
{
   GLdouble aspect = (GLdouble) width /
  height;
   GLdouble left = -2.5, right = 2.5;
   GLdouble bottom = -2.5, top = 2.5;
   glViewport( 0, 0, (GLsizei) w,
  (GLsizei) h );
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
```

... continued ...

resize( ): Ortho (bagian 2)

```
    if ( aspect < 1.0 ) {
       left /= aspect;
       right /= aspect;
    } else {
       bottom *= aspect;
       top *= aspect;
    }
    glOrtho( left, right, bottom, top,
  near, far );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
}
```

Membangun Modeling Transformations

- Masalah 1: hirarki obyek
  - Suatu posisi tergantung pada posisi sebelumnya
  - Lengan robot atau tangan ; *sub-assemblies*
- Solusi 1: memindahkan sistem koordinat lokal
  - modeling transformation untuk memindahkan sistem koordinat
  - post-multiply column-major matrices
  - OpenGL post-multiplies matrices

Membangun Modeling Transformations

- Masalah 2 : obyek berpindah secara relatif pada *absolute world origin*
  - Obyek berotasi di area yang salah pada origin
    - Membuat obyek spin di sekitar center atau suatu area
- Solusi 2: fixed coordinate system
  - modeling transformations akan memindahkan obyek disekitar fixed coordinate system
  - pre-multiply column-major matrices
  - OpenGL post-multiplies matrices
  - harus me-reverse order operasi untuk mendapatkan efek yang diinginkan

Area Clipping Tambahan

- Paling tidak ada 6 atau lebih area clipping
- Baik untuk perhitungan cross-sections
- Modelview matrix memindahkan area clipping

clipped

$$Ax + By + Cz + D < 0$$

```
glEnable( GL_CLIP_PLANEi )
glClipPlane( GL_CLIP_PLANEi, GLdouble* coeff )
```

Reversing Koordinat Proyeksi

- Screen space kembali ke world space

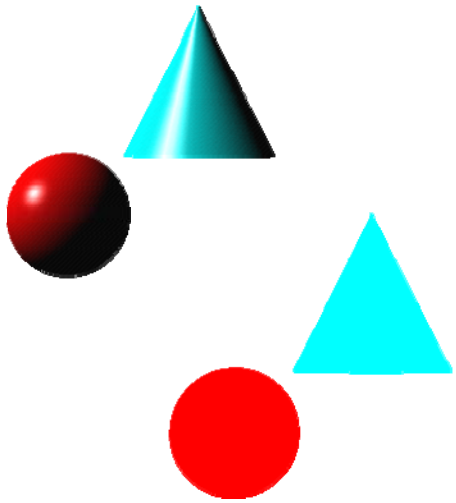
```
glGetIntegerv( GL_VIEWPORT, GLint viewport[4] )
glGetDoublev( GL_MODELVIEW_MATRIX, GLdouble
  mvmatrix[16] )
glGetDoublev( GL_PROJECTION_MATRIX,
                GLdouble projmatrix[16] )
gluUnProject( GLdouble winx, winy, winz,
                mvmatrix[16], projmatrix[16],
                GLint viewport[4],
                GLdouble *objx, *objy, *objz )
```

- `gluProject` untuk memindahkan *world space* ke *screen space*

Pencahayaan (Lighting)

Prinsip Pencahayaan

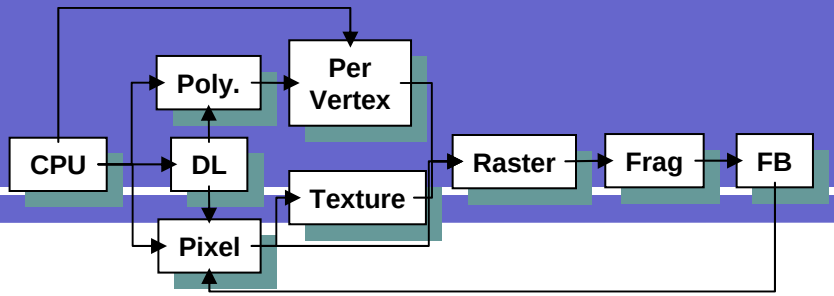
- Pencahayaan mensimulasikan bagaimana obyek memantulkan cahaya
  - Komposisi material obyek
  - Warna cahaya dan posisi
  - Parameter global pencahayaan
    - Cahaya ambien
    - Pencahayaan dua sisi
  - Terdapat pada color index dan mode RGBA



Bagaimana OpenGL Mensimulasikan Cahaya

- Model Pencahayaan Phong
  - Perhitungan pada vertices
- Kontributor Pencahayaan
  - Properti permukaan material
  - Properti cahaya
  - Properti Model pencahayaan

Surface Normals

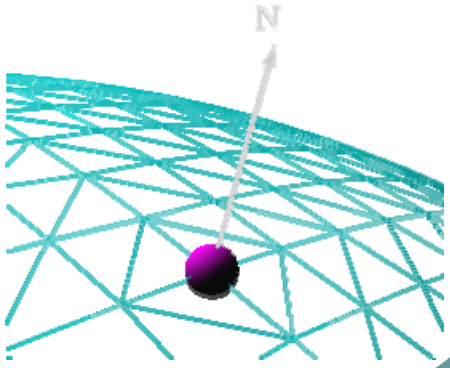


- Normal mendefinisikan bagaimana pemukaan memantulkan cahaya
  - `glNormal3f( x, y, z )`
  - Current normal digunakan untuk menghitung warna vertex
  - Gunakan *unit* normals untuk pencahayaan yang tepat
    - Skalakan efek pada panjang normal

`glEnable( GL_NORMALIZE )`

or

`glEnable( GL_RESCALE_NORMAL )`



Material Properties

- Definisikan properti permukaan obyek primitif

```
glMaterialfv( face, property, value );
```

GL_DIFFUSE	Base color
GL_SPECULAR	Highlight Color
GL_AMBIENT	Low-light Color
GL_EMISSION	Glow Color
GL_SHININESS	Surface Smoothness

  - Pisahkan material antara bagian *front* dan *back*

Properti Cahaya

```
glLightfv( light, property, value );
```

- ***light*** menspesifikasikan jenis cahaya
  - multiple lights, mulai dari **GL_LIGHT0**

    glGetIntegerv(*GL_MAX_LIGHTS, &n*);
- ***properties***
  - Warna
  - posisi dan type
  - Attenuation (metode perambatan)

Sumber cahaya

- Light color properties
  - **GL_AMBIENT**
  - **GL_DIFFUSE**
  - **GL_SPECULAR**

Tipe Cahaya

- OpenGL mendukung 2 tipe Cahaya :
  - Local (Point) light sources
  - Infinite (Directional) light sources
- Tipe cahaya dikendalikan oleh koordinat w

  $w = 0$ ***Infinite Light directed along*** $(x \quad y \quad z)$

  $w \neq 0$ ***Local Light positioned at*** $(x/w \quad y/w \quad z/w)$

Menyalakan (Turning on) Cahaya

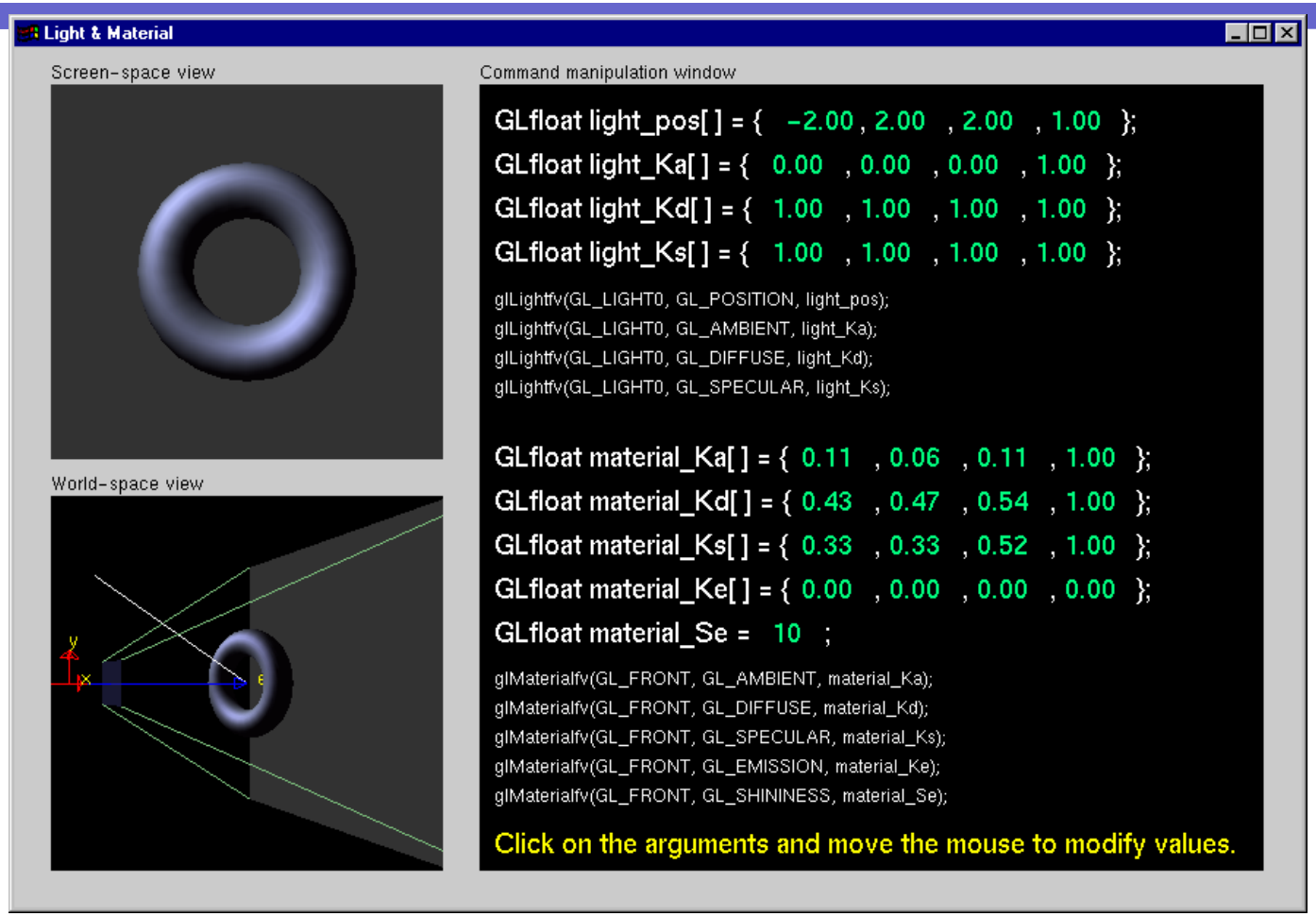
- Flip setiap “switch” cahaya

```
glEnable( GL_LIGHTn );
```

- Turn on the power

```
glEnable( GL_LIGHTING );
```

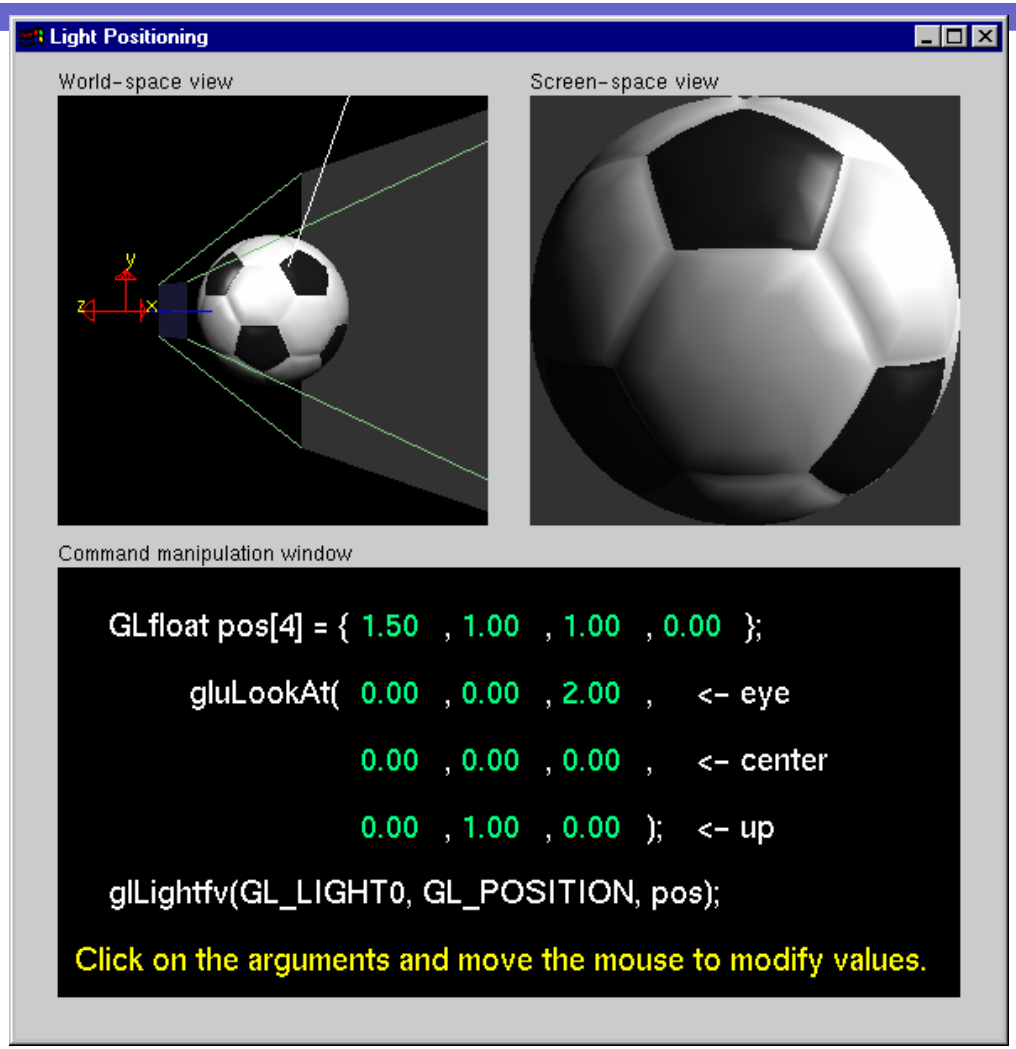
Light Material Tutorial



Mengendalikan Posisi Cahaya

- Modelview matrix berpengaruh pada posisi cahaya:
  - Perbedaan efek berdasarkan kapan posisi cahaya dispesifikasikan
    - eye coordinates
    - world coordinates
    - model coordinates
  - Matrik Push dan pop untuk pengendali unik posisi cahaya

Light Position Tutorial



Fitur Pencahayaan Lanjut

- Spotlight
  - Melokalisasi efek cahaya
    - ***GL_SPOT_DIRECTION***
    - ***GL_SPOT_CUTOFF***
    - ***GL_SPOT_EXPONENT***

Fitur Pencahayaan Lanjut

- Perambatan cahaya (Light attenuation)
  - decrease light intensity with distance
    - ***GL_CONSTANT_ATTENUATION***
    - ***GL_LINEAR_ATTENUATION***
    - ***GL_QUADRATIC_ATTENUATION***

$$f_i = \frac{1}{k_c + k_l d + k_q d^2}$$

Properti Model Cahaya

```
glLightModelfv( property, value );
```

- Enabling two sided lighting

  GL_LIGHT_MODEL_TWO_SIDE
- Global ambient color

  GL_LIGHT_MODEL_AMBIENT
- Local viewer mode

  GL_LIGHT_MODEL_LOCAL_VIEWER
- Separate specular color

  GL_LIGHT_MODEL_COLOR_CONTROL

Tips untuk pencahayaan yg baik

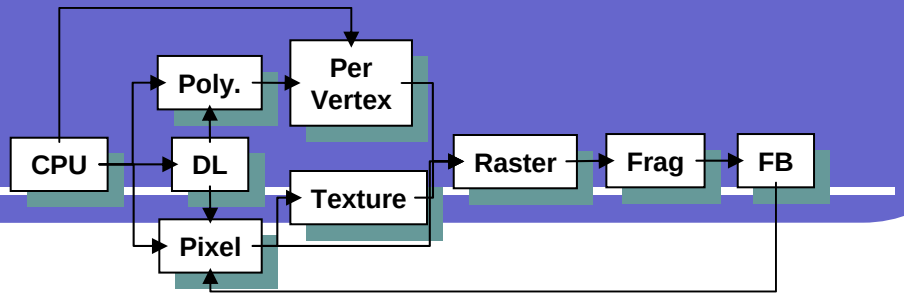
- Pemanggilan Pencahayaan dikomputasi hanya pada vertices
  - Pemanggilan pada model tessellation memang menghasilkan kualitas pencahayaan yang lebih baik tapi proses geometrinya lebih kompleks
- Gunakan cahaya tunggal *infinite* untuk perhitungan pencahayaan cepat
  - Karena komputasi minimal untuk setiap vertex-nya

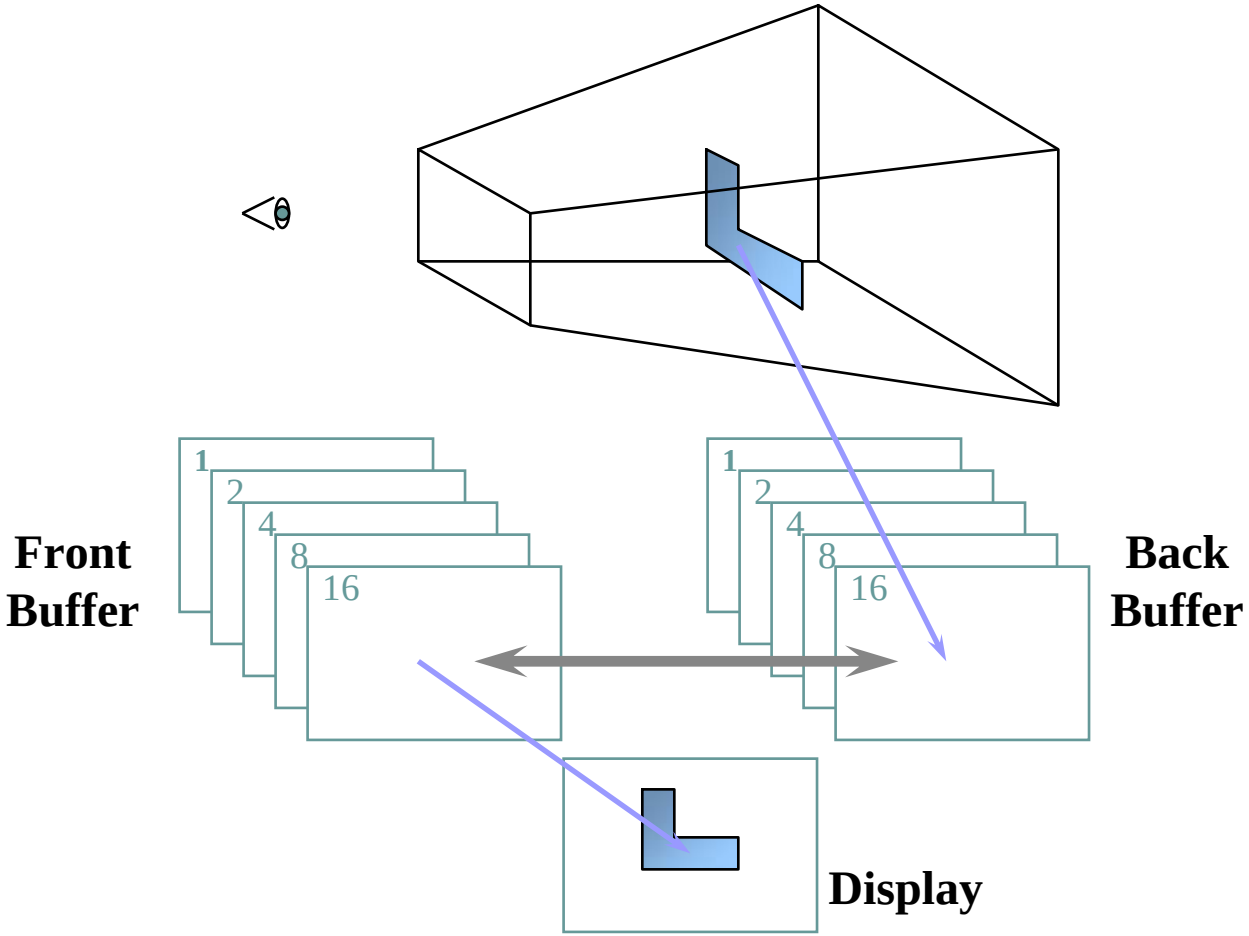
Animasi dan Depth Buffering

Animation dan Depth Buffering

- Mendikusikan double buffering dan animasi
- Mendiskusikan *hidden surface removal* menggunakan *depth buffer*

Double Buffering





Animasi menggunakan Double Buffering

① Defenisikan double buffered pada color buffer

```
glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE
  );
```

② Clear color buffer

```
glClear( GL_COLOR_BUFFER_BIT );
```

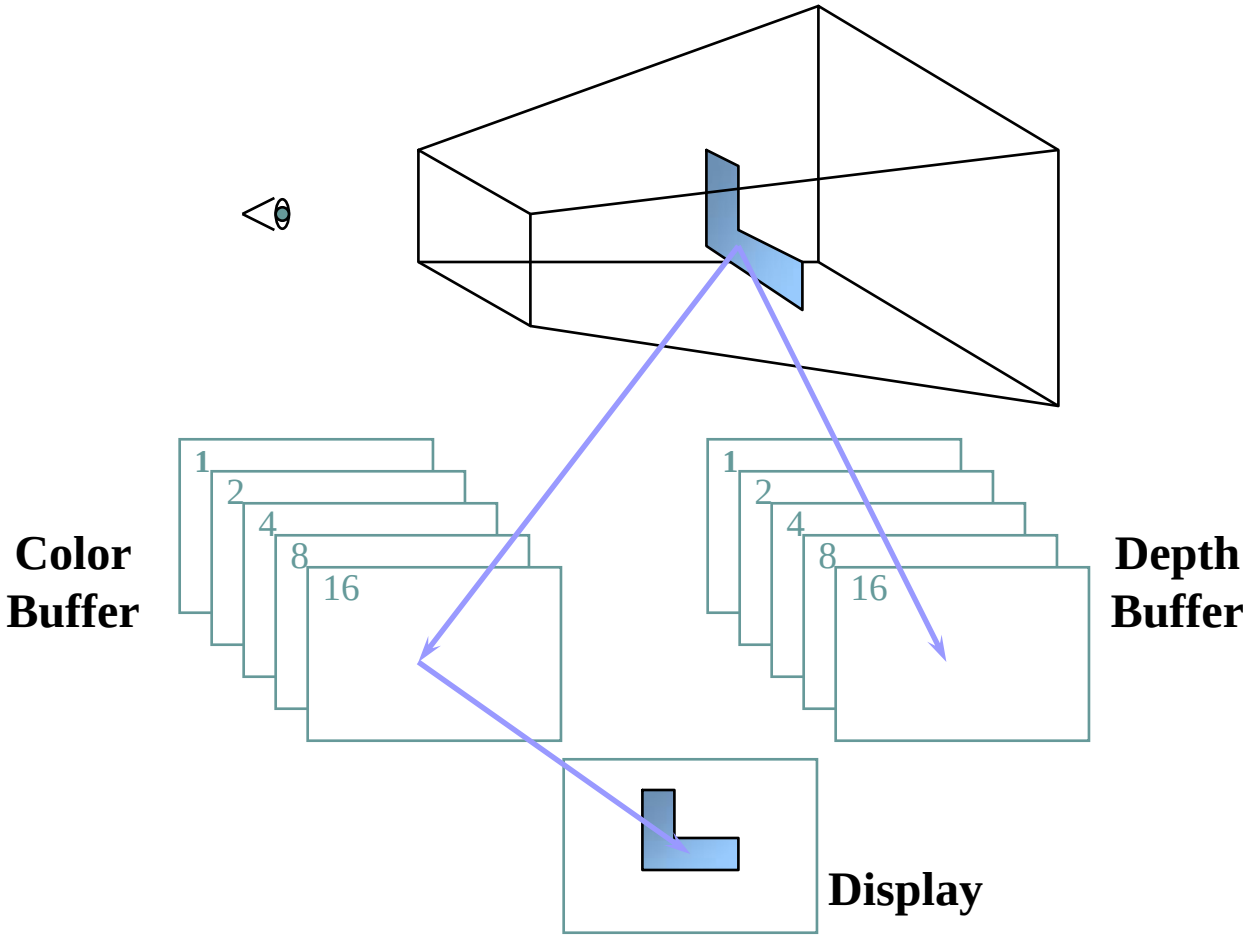
③ Render scene

④ Definisikan swap untuk front dan back buffers

```
glutSwapBuffers();
```

- Ulangi langkah 2 - 4 untuk animasi

Depth Buffering dan Hidden Surface Removal



Depth Buffering menggunakan OpenGL

① Defenisikan depth buffer

```
glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE
  | GLUT_DEPTH );
```

② Enable depth buffering

```
glEnable( GL_DEPTH_TEST );
```

③ Clear color dan depth buffers

```
glClear( GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT );
```

④ Render scene

⑤ Swap color buffers

An Updated Program Template

```
void main( int argc, char** argv )
{
     glutInit( &argc, argv );
     glutInitDisplayMode( GLUT_RGB |
        GLUT_DOUBLE | GLUT_DEPTH );
     glutCreateWindow( “Tetrahedron” );
     init();
     glutIdleFunc( idle );
     glutDisplayFunc( display );
     glutMainLoop();
}
```

An Updated Program Template (Lanjutan)

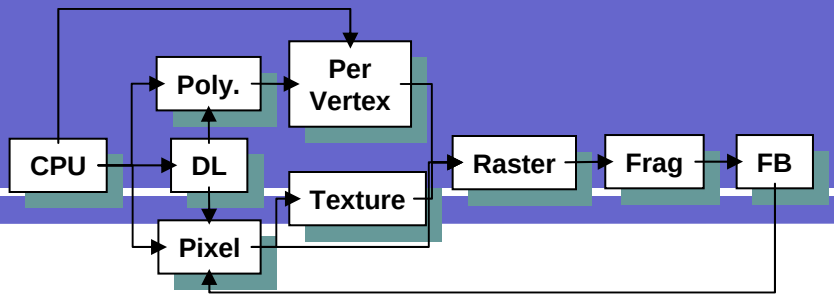
```
void init( void )
  {
     glClearColor( 0.0, 0.0, 1.0,
  1.0 );
  }


  void idle( void )
  {
     glutPostRedisplay();
  }
```

An Updated Program Template (Lanjutan)

```
void drawScene( void )
{
    GLfloat vertices[] = { … };
    GLfloat colors[] = { … };
    glClear( GL_COLOR_BUFFER_BIT |
      GL_DEPTH_BUFFER_BIT );
    glBegin( GL_TRIANGLE_STRIP );
  /* calls to glColor*() and
    glVertex*() */
    glEnd();
    glutSwapBuffers();
}
```

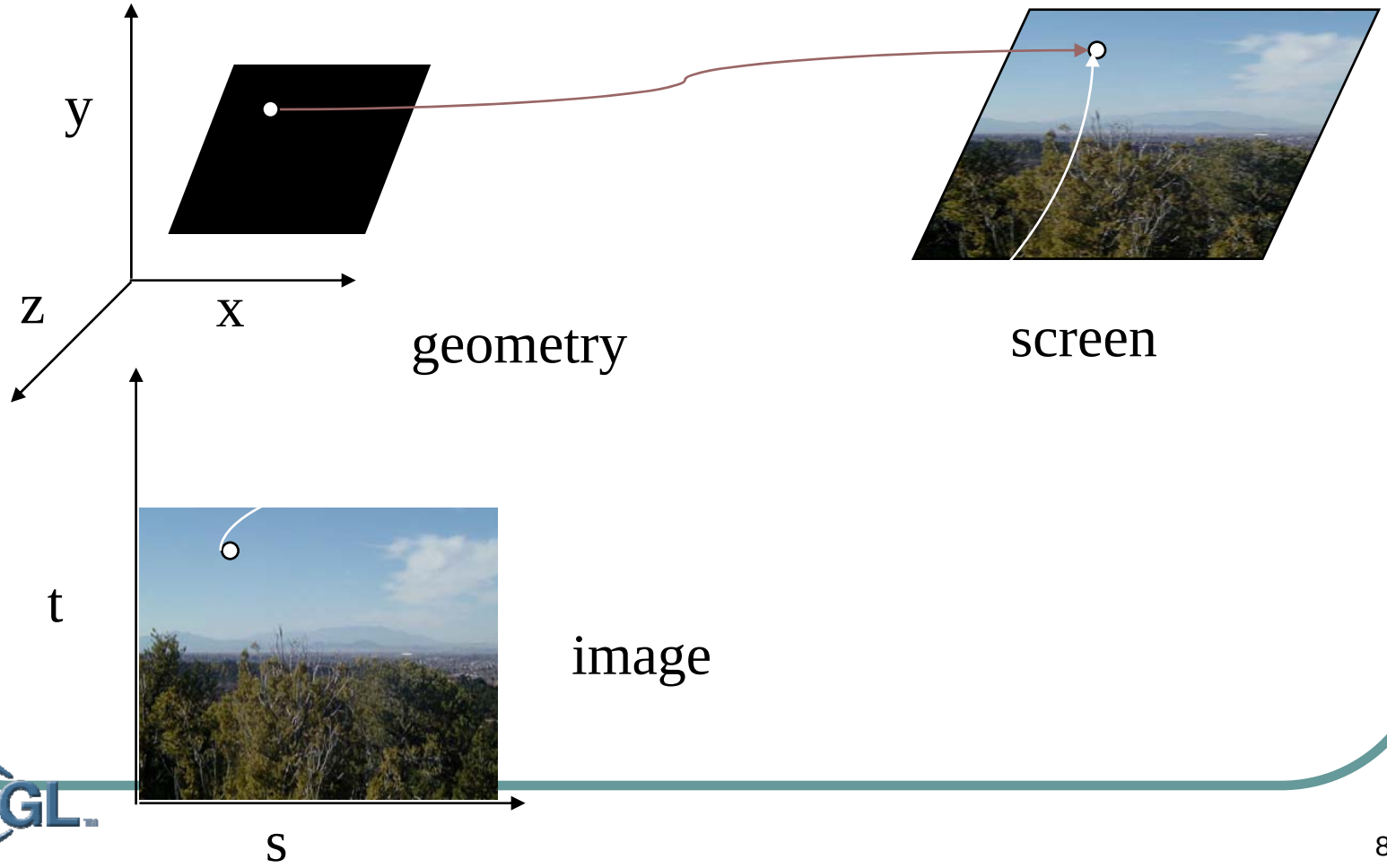
Texture Mapping

Texture Mapping



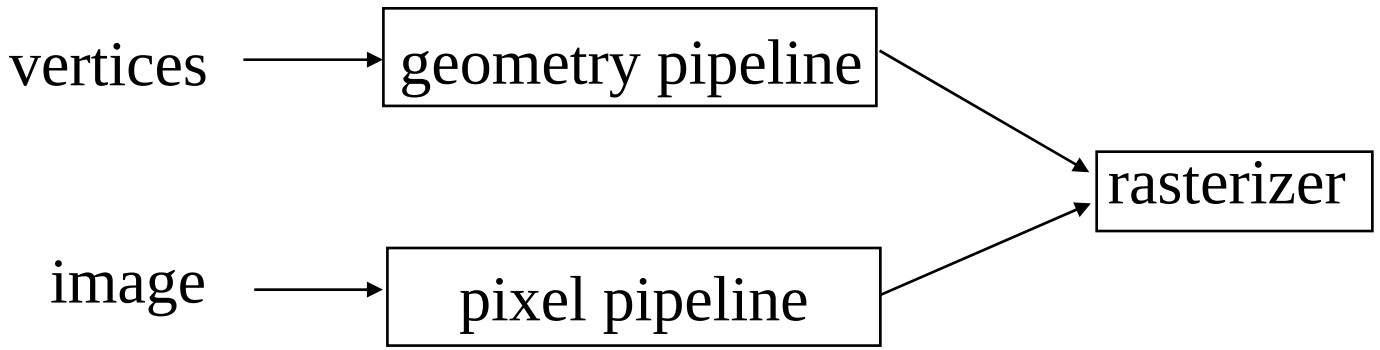
- mengaplikasikan citra 1D, 2D, atau 3D ke geometrik primitif
- Hal yang digunakan untuk proses Texturing
  - simulasi material
  - Mengurangi kompleksitas geometrik
  - image warping
  - refleksi

Texture Mapping



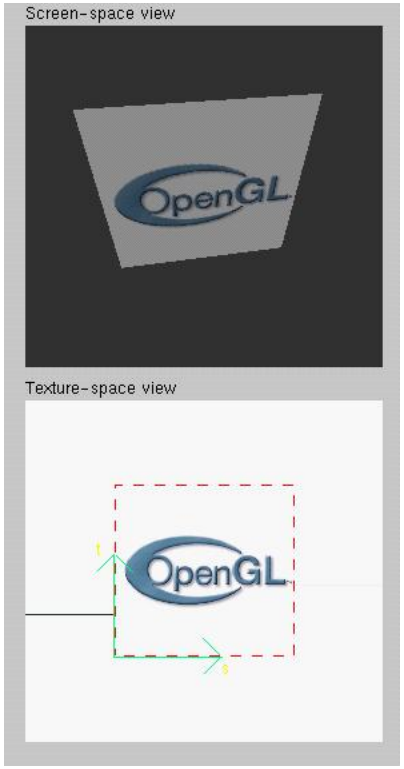
Texture Mapping dan OpenGL Pipeline

- Citra dan geometri mengalir dalam pipeline yang terpisah pada proses di rasterizer
  - “kompleks” textures tidak berpengaruh pada kompleksitas geometrik



Contoh Tekstur

- Tekstur ini adalah citra 256 x 256 yang di gabung (mapped) dengan poligon persegi panjang yang di tampilkan pada perspektif



Cara mengaplikasikan Tekstur I

- Three steps
  1. specify texture
     - read or generate image
     - assign to texture
     - enable texturing
  2. assign texture coordinates to vertices
  3. specify texture parameters
     - wrapping, filtering

Tekstur Obyek

- Like display lists for texture images
  - one image per texture object
  - may be shared by several graphics contexts
- Generate texture names

```
glGenTextures( n, *texIds );
```

Cara mengaplikasikan Tekstur II

- specify textures in texture objects
- set texture filter
- set texture function
- set texture wrap mode
- set optional perspective correction hint
- bind texture object
- enable texturing
- supply texture coordinates for vertex
  - coordinates can also be generated

Tekstur Obyek (lanjutan)

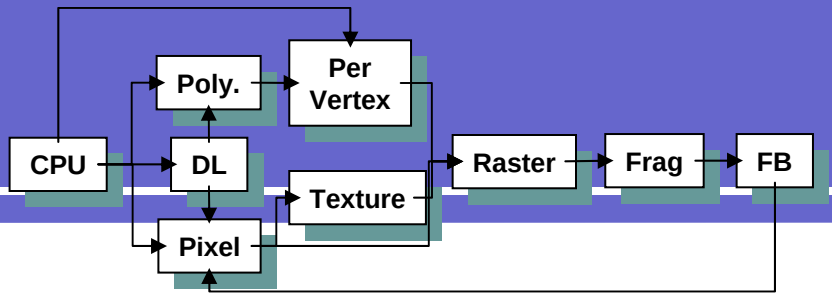
- Create texture objects with texture data and state

```
glBindTexture( target, id );
```

- Bind textures before using

```
glBindTexture( target, id );
```

Memilih citra untuk Tekstur



- Define a texture image from an array of texels in CPU memory
- `glTexImage2D( target, level, components, w, h, border, format, type, *texels );`
  - dimensions of image must be powers of 2
- Texel colors are processed by pixel pipeline
  - pixel scales, biases and lookups can be done

Konversikan citra Tekstur

- If dimensions of image are not power of 2
- `gluScaleImage( format, w_in, h_in, type_in, *data_in, w_out, h_out, type_out, *data_out );`
  - *_in ***is for source image***
  - *_out ***is for destination image***
- Image interpolated and filtered during scaling

Memilih Tekstur: Methode Lainnya

- Use frame buffer as source of texture image
  - uses current buffer as source image

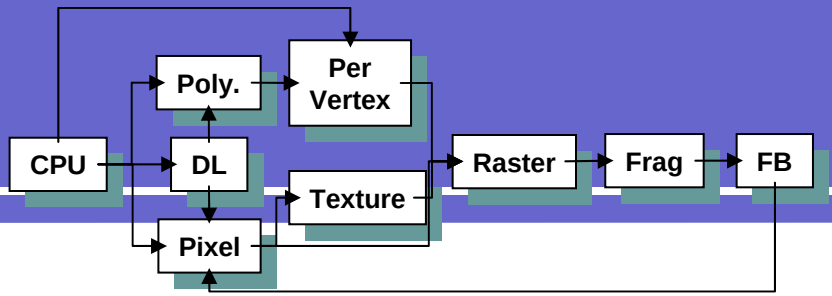
```
glCopyTexImage1D(...)
glCopyTexImage2D(...)
```

- Modify part of a defined texture

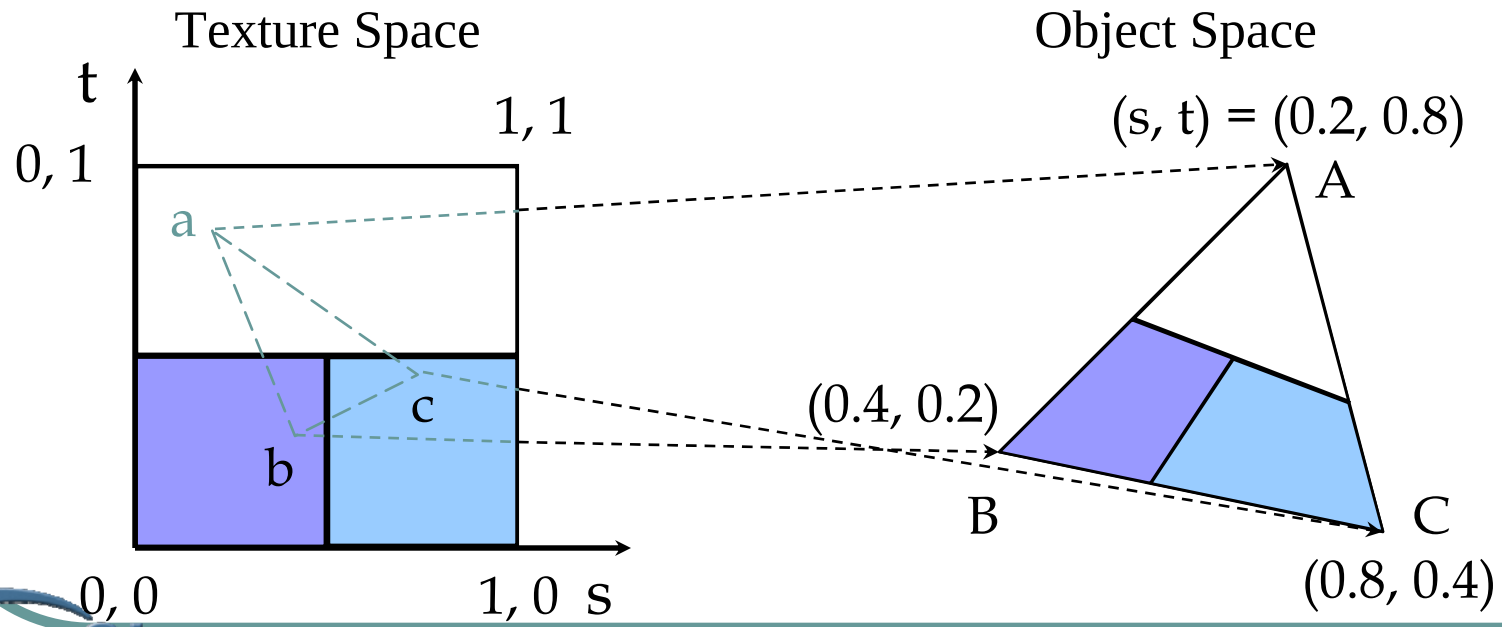
```
glTexSubImage1D(...)
glTexSubImage2D(...)
glTexSubImage3D(...)
```

- Do both with *`glCopyTexSubImage2D(...)`*, etc.

Memetakan (Mapping) Tekstur



- Based on parametric texture coordinates
- `glTexCoord*()` specified at each vertex



Membuat Koordinat Tekstur

- Automatically generate texture coords

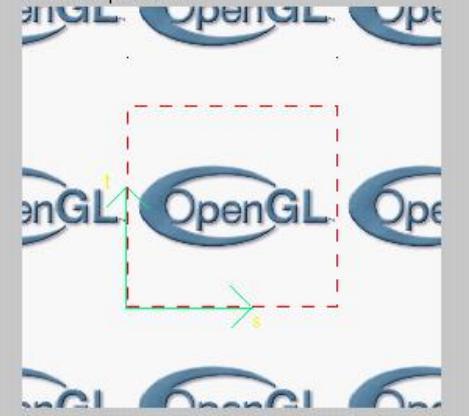
  `glTexGen{ifd}[v]()`

- specify a plane
  - generate texture coordinates based upon distance from plane $Ax + By + Cz + D = 0$
- generation modes
  - `GL_OBJECT_LINEAR`
  - `GL_EYE_LINEAR`
  - `GL_SPHERE_MAP`

Tutorial: Texture

Screen-space view

Texture-space view



Command manipulation window

```
GLfloat border_color[ ] = {  1.00 ,  0.00 ,  0.00 ,  1.00 };
GLfloat env_color[ ] = {     0.00 ,  1.00 ,  0.00 ,  1.00 };

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, env_color);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,  GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,  GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,  GL_REPEAT);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,  GL_MODULATE);

glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);
glColor4f(   0.60  , 0.60  , 0.60  , 1.00  );
glBegin(GL_POLYGON);
glTexCoord2f( 0.0  , 0.0  ); glVertex3f( -1.0 , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 0.0  ); glVertex3f( 1.0  , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 1.0  ); glVertex3f( 1.0  , 1.0  , 0.0  );
glTexCoord2f( 0.0  , 1.0  ); glVertex3f( -1.0 , 1.0  , 0.0  );
glEnd();
```

Click on the arguments and move the mouse to modify values.

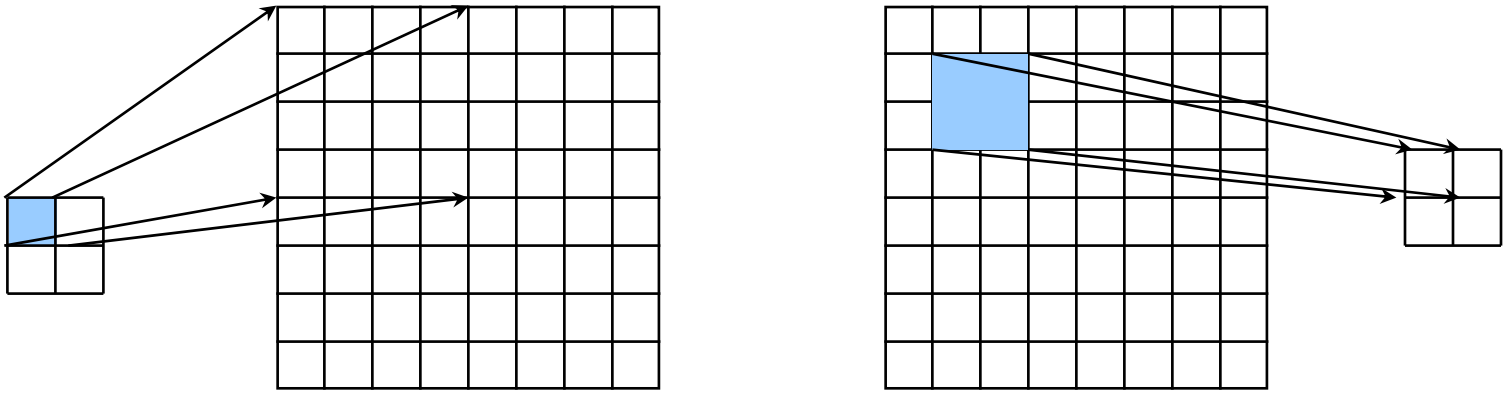
Metode Aplikasi Tekstur

- Filter Modes
  - minification or magnification
  - special mipmap minification filters
- Wrap Modes
  - clamping or repeating
- Texture Functions
  - how to mix primitive's color with texture's color
    - blend, modulate or replace texels

Filter Modes

Example:

```
glTexParameteri( target, type, mode );
```



Texture Polygon

Magnification

Texture Polygon

Minification

Tekstur Mipmapped

- Mipmap allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition

```
glTexImage*D( GL_TEXTURE_*D, level, … )
```

- GLU mipmap builder routines

```
gluBuild*DMipmaps( … )
```

- OpenGL 1.2 introduces advanced LOD controls

Mode Wrapping

- Example:

```
glTexParameteri( GL_TEXTURE_2D,
      GL_TEXTURE_WRAP_S, GL_CLAMP )
glTexParameteri( GL_TEXTURE_2D,
      GL_TEXTURE_WRAP_T, GL_REPEAT )
```

Fungsi berkaitan dengan Tekstur

- Controls how texture is applied
  - `glTexEnv{fi}[v](` *`GL_TEXTURE_ENV, prop, param`* `)`
- *`GL_TEXTURE_ENV_MODE`* modes
  - **`GL_MODULATE`**
  - **`GL_BLEND`**
  - **`GL_REPLACE`**
- Set blend color with *`GL_TEXTURE_ENV_COLOR`*

Tips untuk Koreksi Perspektif

- Texture coordinate and color interpolation
  - either linearly in screen space
  - or using depth/perspective values (slower)
- Noticeable for polygons “on edge”
  - `glHint( GL_PERSPECTIVE_CORRECTION_HINT, hint )`

where ***hint*** is one of

- ***GL_DONT_CARE***
- ***GL_NICEST***
- ***GL_FASTEST***

Adakah tempat untuk Tekstur?

- Query largest dimension of texture image
  - typically largest square texture
  - doesn't consider internal format size
- `glGetIntegerv( GL_MAX_TEXTURE_SIZE, &size )`
- Texture proxy
  - will memory accommodate requested texture size?
  - no image specified; placeholder
  - if texture won't fit, texture state variables set to 0
    - doesn't know about other textures
    - only considers whether this one texture will fit all of memory

Texture Residency

- Working set of textures
  - high-performance, usually hardware accelerated
  - textures must be in texture objects
  - a texture in the *working set* is *resident*
  - for residency of current texture, check **`GL_TEXTURE_RESIDENT`** state
- If too many textures, not all are resident
  - can set priority to have some kicked out first
  - establish 0.0 to 1.0 priorities for texture objects

Imaging and Raster Primitives

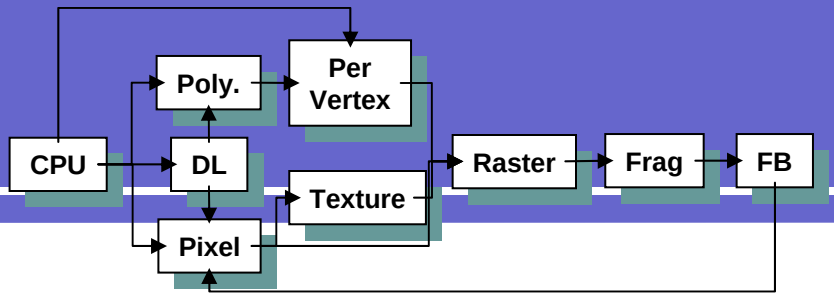
Imaging and Raster Primitives

- Describe OpenGL's raster primitives: bitmaps and image rectangles
- Demonstrate how to get OpenGL to read and render pixel rectangles

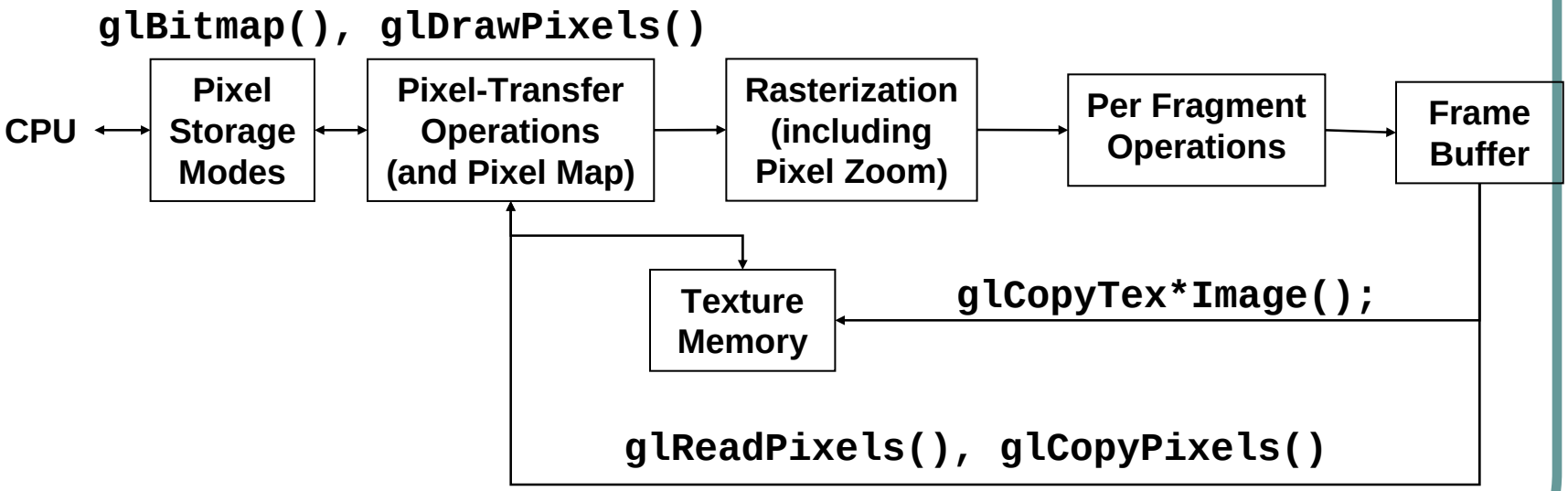
Pixel-based primitives

- Bitmaps
  - 2D array of bit masks for pixels
    - update pixel color based on current color
- Images
  - 2D array of pixel color information
    - complete color information for each pixel
- OpenGL doesn’t understand image formats

Pixel Pipeline



- Programmable pixel storage and transfer operations



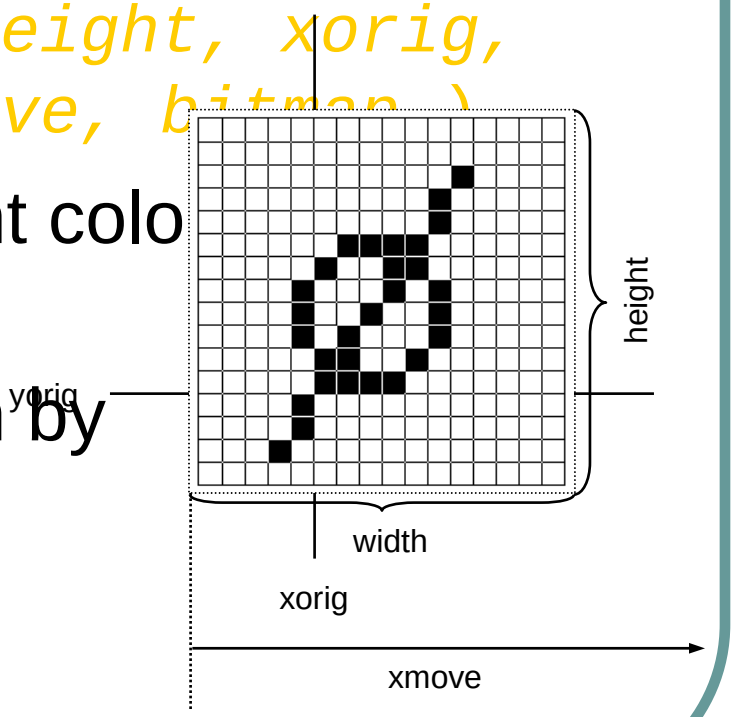
Positioning Image Primitives

- `glRasterPos3f( x, y, z )`
- raster position transformed like geometry
- discarded if raster position is outside of viewport
  - may need to fine tune viewport for desired results

Raster Position

Rendering Bitmaps

- `glBitmap( width, height, xorig, yorig, xmove, ymove, bitmap )`
- render bitmap in current color at $(x - xorig, y - yorig)$
- advance raster position by $(xmove, ymove)$ after rendering



Rendering Fonts using Bitmaps

- OpenGL uses bitmaps for font rendering
  - each character is stored in a display list containing a bitmap
  - window system specific routines to access system fonts
    - **`glXUseXFont()`**
    - **`wglUseFontBitmaps()`**

Rendering Images

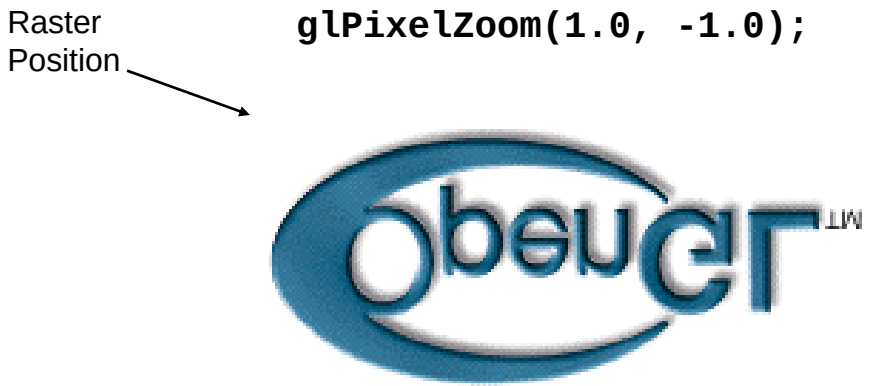
- `glDrawPixels( width, height, format, type, pixels )`
  - render pixels with lower left of image at current raster position
  - numerous formats and data typ for specifying storage in memory
    - best performance by using format and type that matches hardware

Reading Pixels

- `glReadPixels( x, y, width, height, format, type, pixels )`
  - read pixels from specified (x,y) position in framebuffer
  - pixels automatically converted from framebuffer format into requested format and type
- Framebuffer pixel copy
  - `glCopyPixels( x, y, width, height, type )`

Pixel Zoom

- glPixelZoom(*x, y*)
  - expand, shrink or reflect pixels around current raster position
  - fractional zoom supported



Storage and Transfer Modes

- Storage modes control accessing memory
  - byte alignment in host memory
  - extracting a subimage
- Transfer modes allow modify pixel values
  - scale and bias pixel component values
  - replace colors using pixel maps

Advanced OpenGL Topics

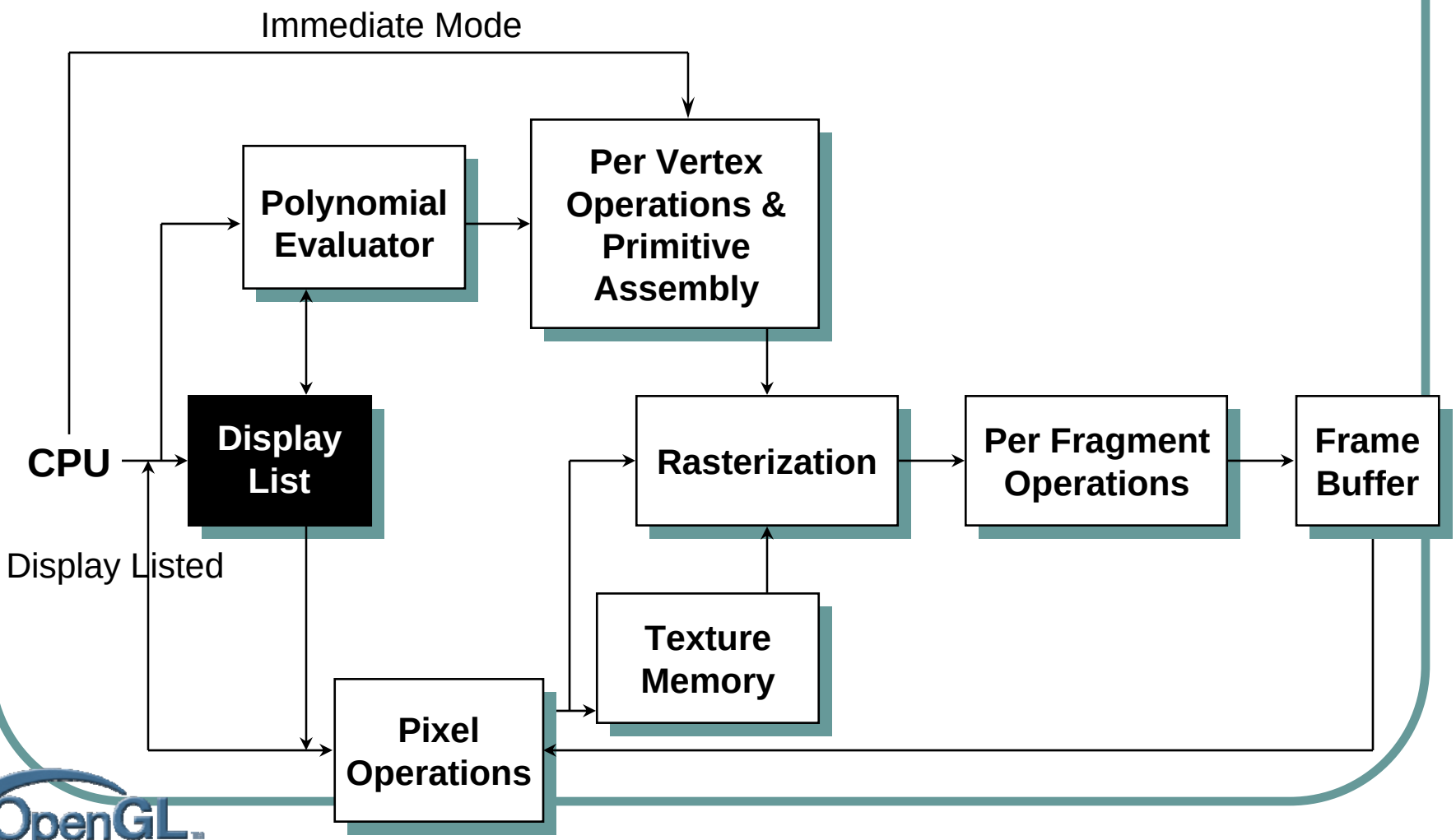
Advanced OpenGL Topics

- Display Lists and Vertex Arrays
- Alpha Blending and Antialiasing
- Using the Accumulation Buffer
- Fog
- Feedback & Selection
- Fragment Tests and Operations
- Using the Stencil Buffer

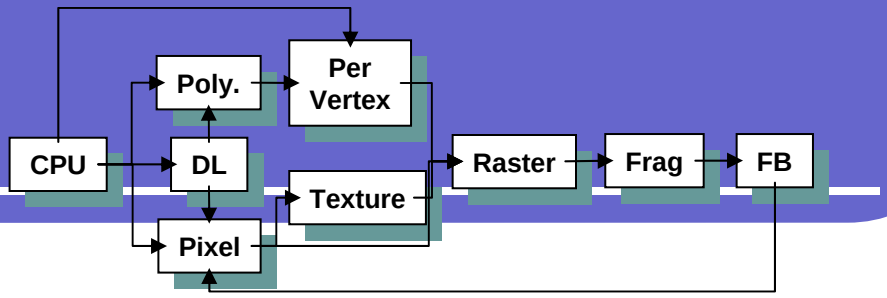
Immediate Mode versus Display Listed Rendering

- 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

Immediate Mode versus Display Lists



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 );
}
```

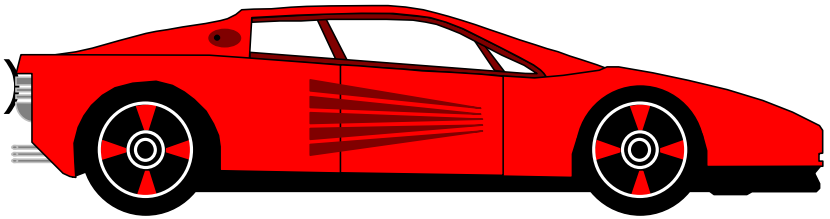
Display Lists

- Not all OpenGL routines can be stored in display lists
- State changes persist, even after a display list is finished
- Display lists can call other display lists
- Display lists are not editable, but you can fake it
  - make a list (A) which calls other lists (B, C, and D)
  - delete and replace B, C, and D, as needed

Display Lists and Hierarchy

- Consider model of a car
  - Create display list for chassis
  - Create display list for wheel

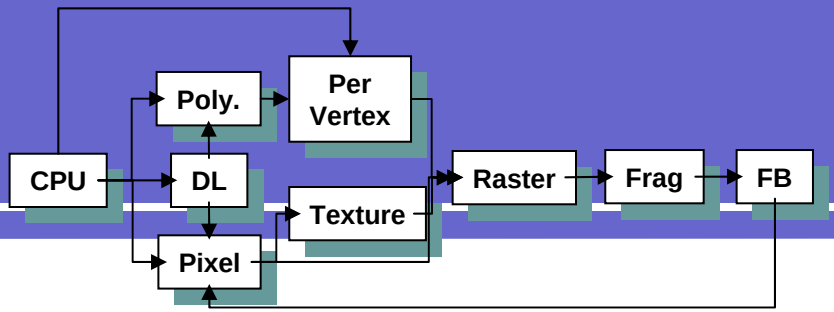
```
glNewList( CAR, GL_COMPILE );
    glCallList( CHASSIS );
    glTranslatef( … );
    glCallList( WHEEL );
    glTranslatef( … );
    glCallList( WHEEL );
            …
glEndList();
```



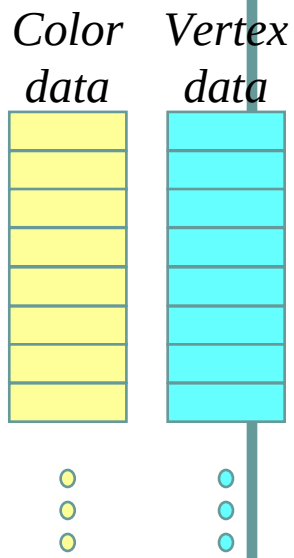
Advanced Primitives

- Vertex Arrays
- Bernstein Polynomial Evaluators
  - basis for GLU NURBS
    - NURBS (Non-Uniform Rational B-Splines)
- GLU Quadric Objects
  - sphere
  - cylinder (or cone)
  - disk (circle)

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



Why use Display Lists or Vertex Arrays?

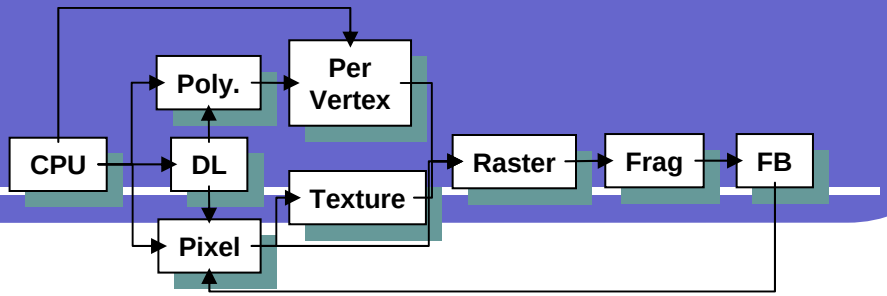
- May provide better performance than immediate mode rendering
- Display lists can be shared between multiple OpenGL context
  - reduce memory usage for multi-context applications
- Vertex arrays may format data for better memory access

Alpha: the 4th Color Component

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

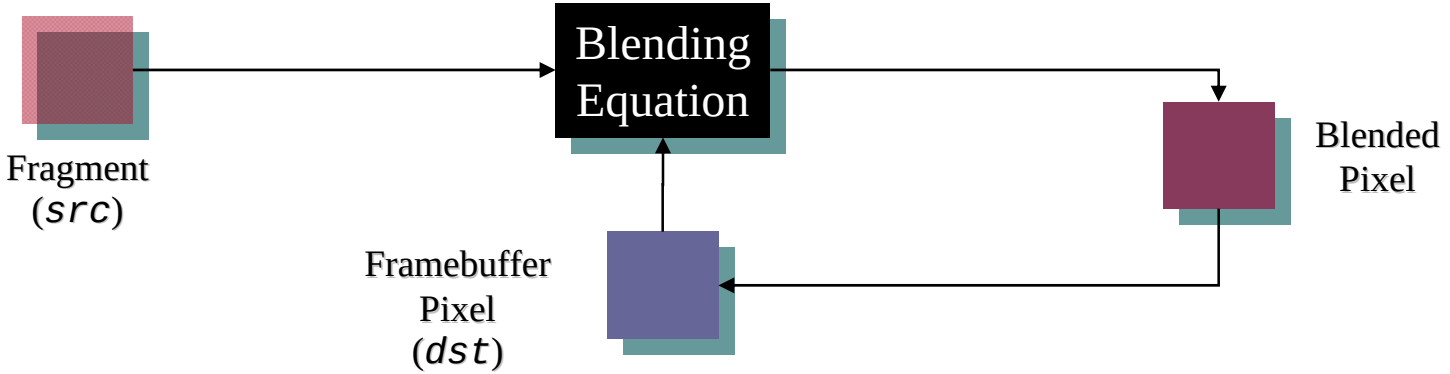
```
glEnable( GL_BLEND )
```

Blending



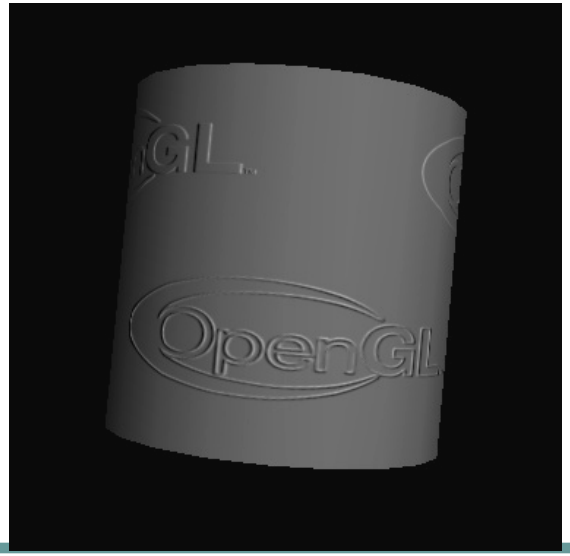
- Combine pixels with what's in already in the framebuffer
  - `glBlendFunc( src, dst )`

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



Multi-pass Rendering

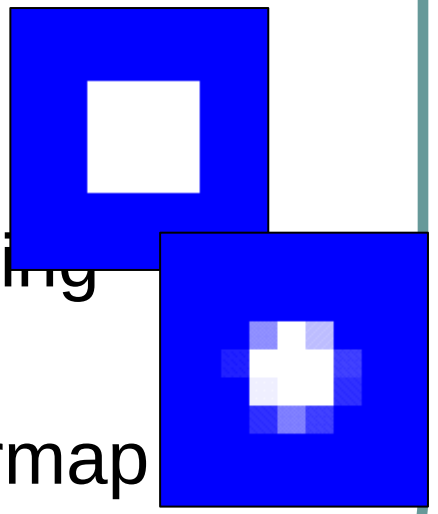
- Blending allows results from multiple drawing passes to be combined together
  - enables more complex rendering algorithms



Example of bump-mapping done with a multi-pass OpenGL algorithm

Antialiasing

- Removing the Jaggies
  - `glEnable( mode )`
    - **`GL_POINT_SMOOTH`**
    - **`GL_LINE_SMOOTH`**
    - **`GL_POLYGON_SMOOTH`**
  - alpha value computed by computing sub-pixel coverage
  - available in both RGBA and colormap modes



Accumulation Buffer

- Problems of compositing into color buffers
  - limited color resolution
    - clamping
    - loss of accuracy
  - Accumulation buffer acts as a “floating point” color buffer
    - accumulate into accumulation buffer
    - transfer results to frame buffer

Accessing Accumulation Buffer

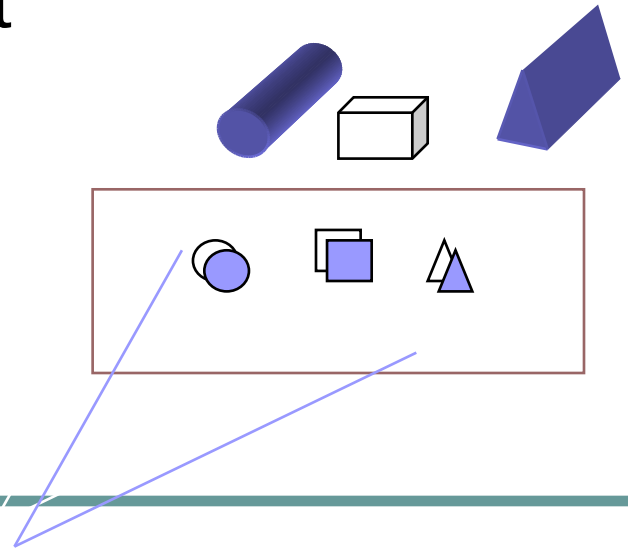
- `glAccum( op, value )`
- operations
  - within the accumulation buffer: ***GL_ADD, GL_MULT***
  - from read buffer: ***GL_ACCUM, GL_LOAD***
  - transfer back to write buffer: ***GL_RETURN***
- **`glAccum(GL_ACCUM, 0.5)`** multiplies each value in write buffer by 0.5 and adds to accumulation buffer

Accumulation Buffer Applications

- Compositing
- Full Scene Antialiasing
- Depth of Field
- Filtering
- Motion Blur

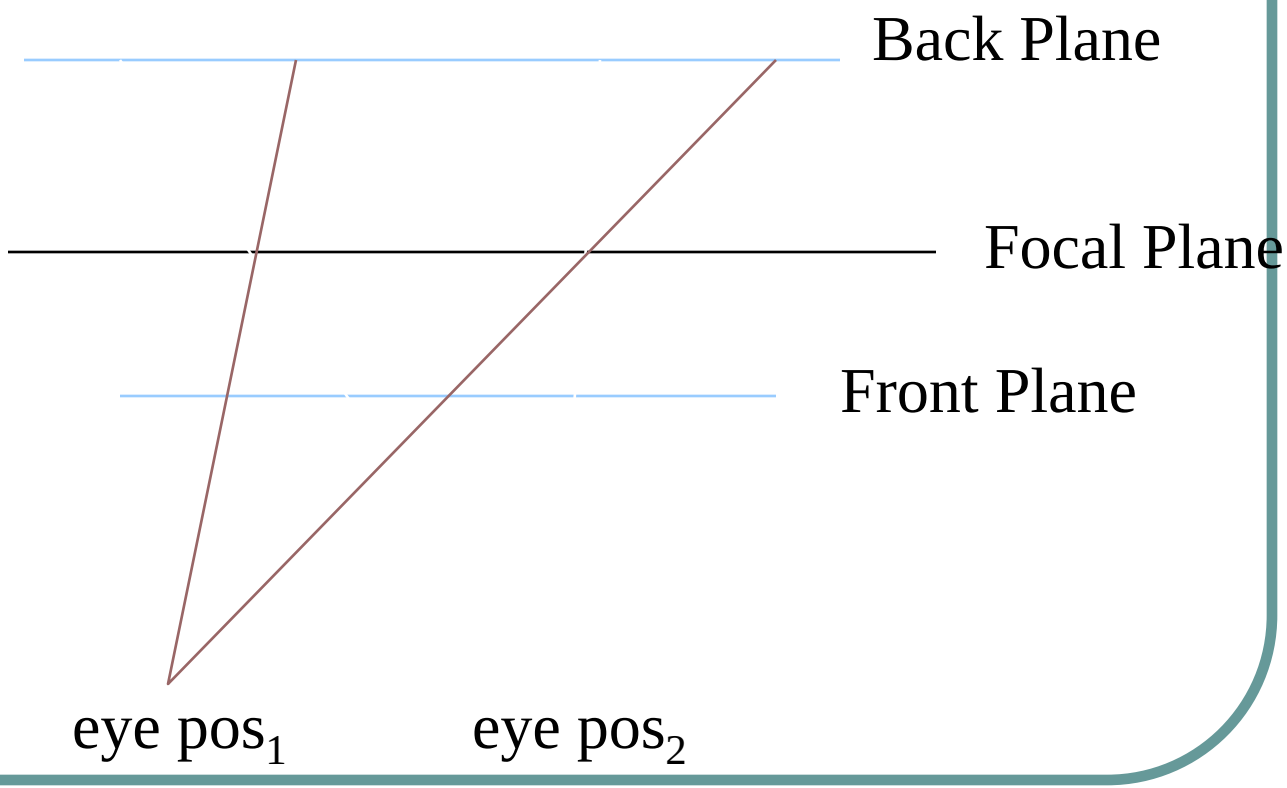
Full Scene Antialiasing : *Jittering the view*

- Each time we move the viewer, the image shifts
  - Different aliasing artifacts in each image
  - Averaging images using accumulation buffer averages out these artifacts



Depth of Focus : *Keeping a Plane in Focus*

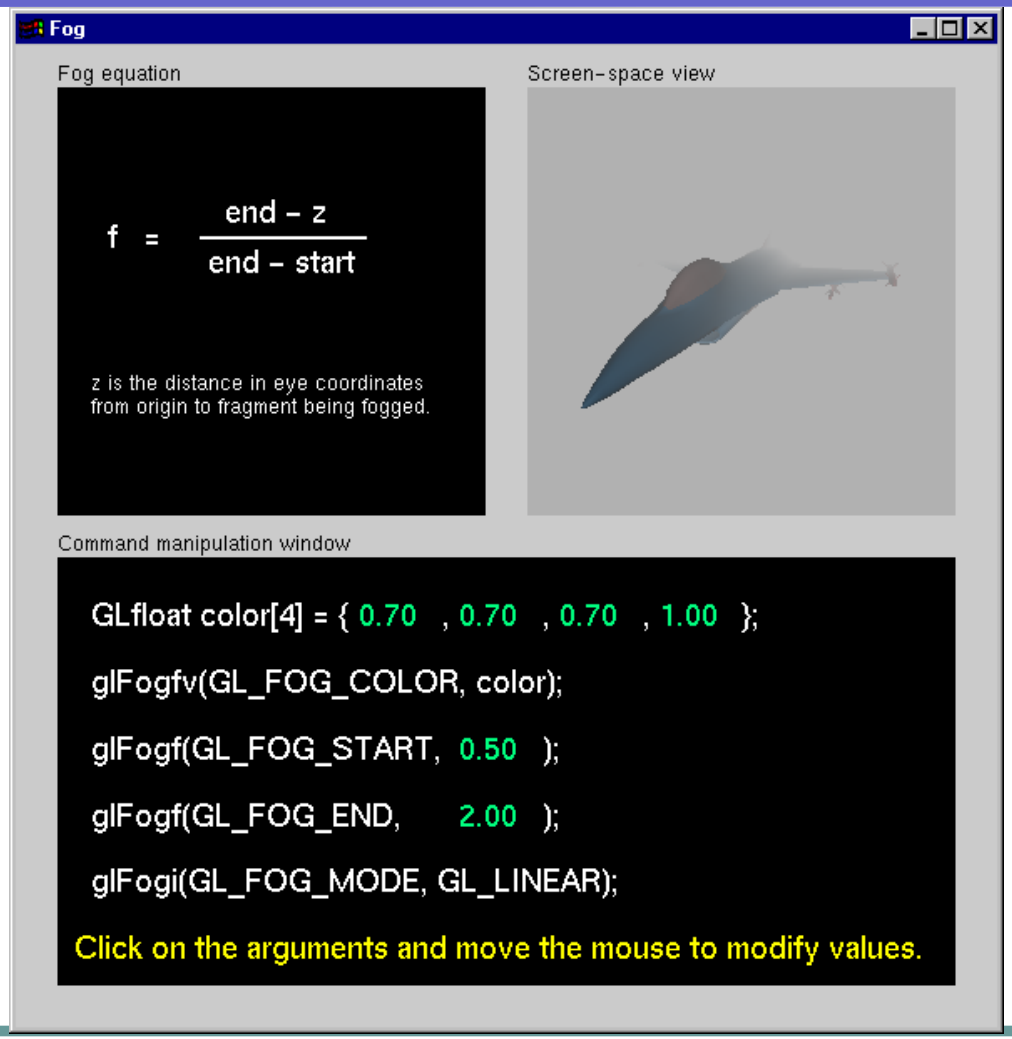
- Jitter the viewer to keep one plane unchanged



Fog

- `glFog{if}( property, value )`
- Depth Cueing
  - Specify a range for a linear fog ramp
    - **`GL_FOG_LINEAR`**
- Environmental effects
  - Simulate more realistic fog
    - **`GL_FOG_EXP`**
    - **`GL_FOG_EXP2`**

Fog Tutorial



Feedback Mode

- Transformed vertex data is returned to the application, not rendered
  - useful to determine which primitives will make it to the screen
- Need to specify a feedback buffer
- `glFeedbackBuffer( size, type, buffer )`
- Select feedback mode for rendering
  - `glRenderMode( GL_FEEDBACK )`

Selection Mode

- Method to determine which primitives are inside the viewing volume
- Need to set up a buffer to have results returned to you

```
glSelectBuffer( size, buffer )
```

- Select selection mode for rendering
  - `glRenderMode( GL_SELECT )`

Selection Mode (cont.)

- To identify a primitive, give it a name
  - “names” are just integer values, not strings
- Names are stack based
  - allows for hierarchies of primitives
- Selection Name Routines

```
glLoadName( name ) glPushName( name )
glInitNames()
```

Picking

- Picking is a special case of selection
- Programming steps
  - restrict “drawing” to small region near pointer

    use **`gluPickMatrix()`** on projection matrix
  - enter selection mode; re-render scene
  - primitives drawn near cursor cause hits
  - exit selection; analyze hit records

Picking Template

```
glutMouseFunc( pickMe );

void pickMe( int button, int state, int x, int y )
   {
    GLuint nameBuffer[256];
    GLint hits;
    GLint myViewport[4];
    if (button != GLUT_LEFT_BUTTON ||
        state != GLUT_DOWN) return;
    glGetIntegerv( GL_VIEWPORT, myViewport );
    glSelectBuffer( 256, nameBuffer );
    (void) glRenderMode( GL_SELECT );
    glInitNames();
```

Picking Template (cont.)

```
   glMatrixMode( GL_PROJECTION );
   glPushMatrix();
   glLoadIdentity();
   gluPickMatrix( (GLdouble) x, (GLdouble)
      (myViewport[3]-y), 5.0, 5.0, myViewport );
/*   gluPerspective or glOrtho or other
  projection  */
   glPushName( 1 );
/*   draw something  */
   glLoadName( 2 );
/*   draw something else … continue …  */
```

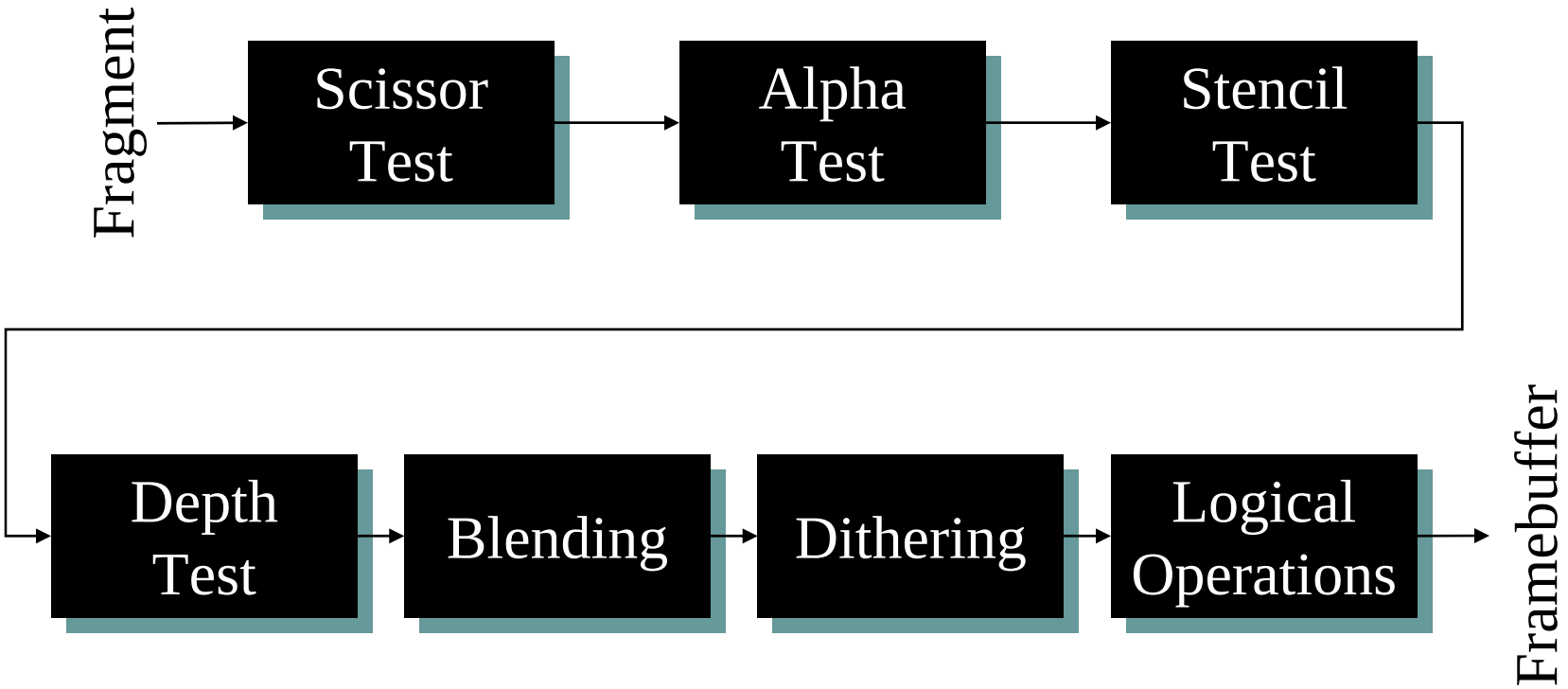
Picking Template (cont.)

```
    glMatrixMode( GL_PROJECTION );
    glPopMatrix();
    hits = glRenderMode( GL_RENDER );
/*    process nameBuffer  */
}
```

Picking Ideas

- For OpenGL Picking Mechanism
  - only render what is pickable (e.g., don't clear screen!)
  - use an "invisible" filled rectangle, instead of text
  - if several primitives drawn in picking region, hard to use z values to distinguish which primitive is "on top"
- Alternatives to Standard Mechanism
  - color or stencil tricks (for example, use **`glReadPixels()`** to obtain pixel value from back buffer)

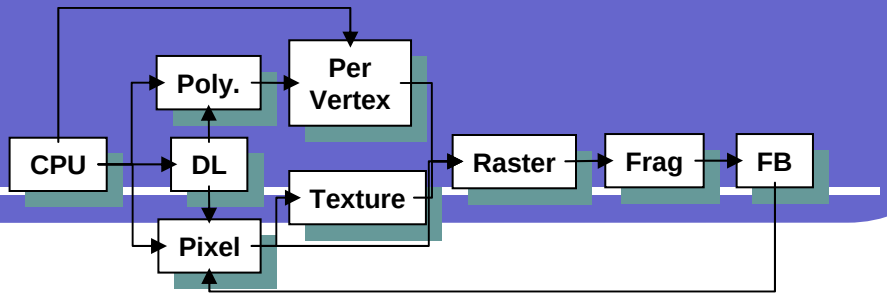
Getting to the Framebuffer



Scissor Box

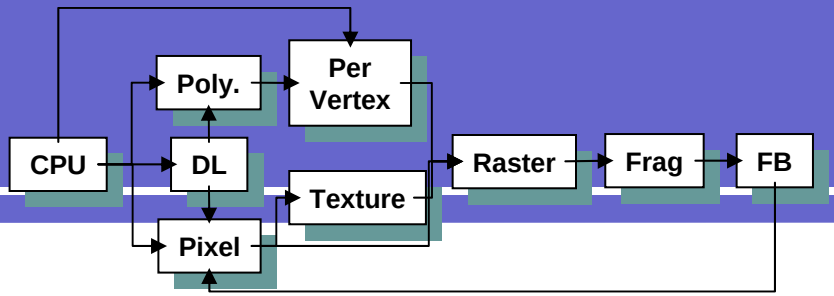
- Additional Clipping Test
  - `glScissor( x, y, w, h )`
  - any fragments outside of box are clipped
  - useful for updating a small section of a viewport
    - affects **`glClear()`** operations

Alpha Test

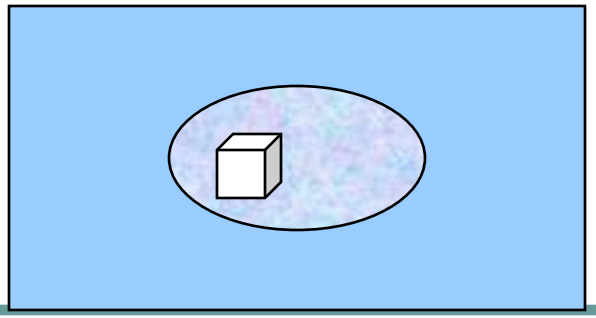


- Reject pixels based on their alpha value
  - `glAlphaFunc( func, value )`
    - `glEnable( GL_ALPHA_TEST )`
  - use alpha as a mask in textures

Stencil Buffer



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



Controlling Stencil Buffer

- `glStencilFunc( func, ref, mask )`
  - compare value in buffer with **`ref`** using **`func`**
  - only applied for bits in **`mask`** which are 1
  - **`func`** is one of standard comparison functions
- `glStencilOp( fail, zfail, zpass )`
  - Allows changes in stencil buffer based on passing or failing stencil and depth tests: **`GL_KEEP`**, **`GL_INCR`**

Creating a Mask

- `glInitDisplayMode( …|GLUT_STENCIL|… );`
- `glEnable( GL_STENCIL_TEST );`
- `glClearStencil( 0x0 );`

- `glStencilFunc( GL_ALWAYS, 0x1, 0x1 );`
- `glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );`
- *draw mask*

Using Stencil Mask

- Draw objects where stencil = 1
  - `glStencilFunc( GL_EQUAL, 0x1, 0x1 )`
- Draw objects where stencil != 1
  - `glStencilFunc( GL_NOTEQUAL, 0x1, 0x1 );`
    - `glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );`

Dithering

- `glEnable( GL_DITHER )`
- Dither colors for better looking results
  - Used to simulate more available colors

Logical Operations on Pixels

- Combine pixels using bitwise logical operations
  - `glLogicOp( mode )`
  - Common modes
    - **GL_XOR**
    - **GL_AND**

Advanced Imaging

- Imaging Subset
  - Only available if **`GL_ARB_imaging`** defined
    - Color matrix
    - Convolutions
    - Color tables
    - Histogram
    - MinMax
    - Advanced Blending

Ringkasan dan Penutup

On-Line Resources

- **http://www.opengl.org**
  - start here; up to date specification and lots of sample code
- **news:comp.graphics.api.opengl**
- **http://www.sgi.com/software/opengl**
- **http://www.mesa3d.org/**
  - Brian Paul's Mesa 3D
- **http://www.cs.utah.edu/~narobins/opengl.html**
  - very special thanks to Nate Robins for the OpenGL Tutors
  - source code for tutors available here!

Buku

- OpenGL Programming Guide, 3rd Edition
- OpenGL Reference Manual, 3rd Edition
- OpenGL Programming for the X Window System
  - includes many GLUT examples
- Interactive Computer Graphics: A top-down approach with OpenGL, 2nd Edition