Computer Graphics: Raster Graphics, Image Formation, Color,2D Viewing, First Graphics Program

Gerda Kamberova

Overview

- Graphics Architecture
- Raster Graphics
- Image formation
  - Human visual system
  - Pinhole camera model
  - Synthetic camera
  - Graphics API (object-viewer-projection)
- Color
  - Physics of color
  - 3-color theory and display technology
  - Human color perception
  - RGB color model in CG
  - Indexed color model

Overview (cont)

- 2D Viewing
  - Clipping window
  - Viewports
  - Window to viewport transformation
- Simple graphics program

Graphics Standard

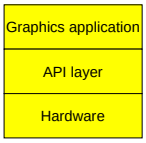
- Goal: portability, standard graphics functions are independent of programming language, bindings are defined for various languages
- GKS, PHIGS, OpenGL, Java3D
- The standard provides specifications for basic graphics functions. To insure independence and portability, there is no standard for graphics interface to output devices, image formats, or transfer protocols

Creating graphics and animation

- **One way to create computer graphics is to use some higher level ready made package**
  - Maya Alias|Wavefront
  - 3ds Max (http://www.discreet.com)
  - 3D Blender (http://www.blender3d.com/)
- **The alternative approach is to "Do it yourself".** This is particularly useful in the design of special applications, e.g. writing computer game engines and scientific visualization packages. It helps to use an Application Programming Interface (API), for example OpenGL, or Direct3D, or Java 3D. APIs are used with a high level programming language.
- Often even "Do it yourself" graphics involves objects created with high level ready made packages, for example, the characters in a game may be designed with a modeling package and then the meshes are imported in the custom designed game engine.

APIs and Device Independent Graphics Programming

- **Using APIs makes it possible to create programs that can be compiled and run on many different graphics platforms.** Most APIs are basically libraries of routines that take care of standard tasks like rasterization, hidden surface removal, polygon clipping, projections.



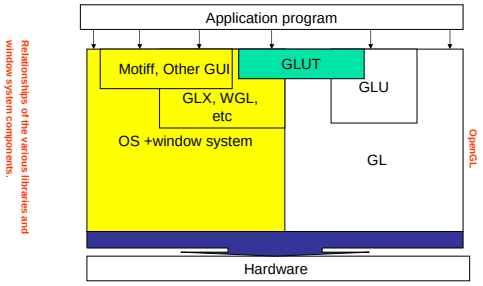
The APIs serve as intermediaries between the application layer and the hardware layer. One of their main functions is to facilitate rendering but they can provide also higher level as well as lower level functions.

Different API levels

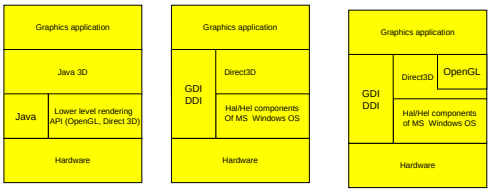
- Lower level APIs (e.g. the rendering APIs Dircect3D and OpenGL) deal directly with the rendering task.
- On the other hand a higher level API like the scene graph API Java3D takes care of maintaining and organizing information about the graphic scenes, illumination, and environment states but needs the help a rendering API to complete the rendering task.

The relationship between different APIs, the operating system, and the hardware layer are illustrated in the following two pages.

Application Programs, OpenGL, OS, Hardware and Drivers

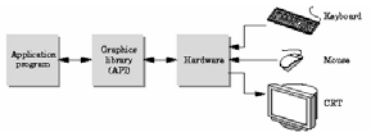


Application Programs, Java 3D, Direct3D, OpenGL, Hardware and Drivers



Graphics Standard

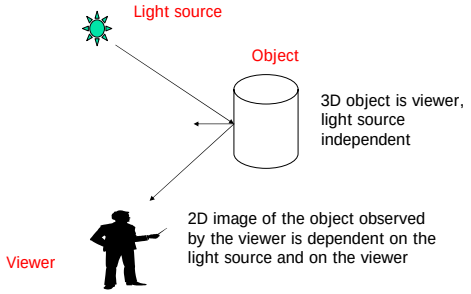
- General purpose API
  - For use in high level programming language
  - By programmers



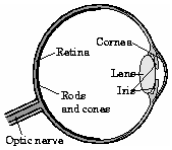
Graphics Standard

- API contains **functions** for creation, manipulation and display of images
  - Basic building blocks: **primitives and attributes**
  - **Geometric transformations**
  - **Modeling transformations**
  - **Viewing transformations**
  - **Structuring /** manipulating components **hierarchically**
  - **Control functions**

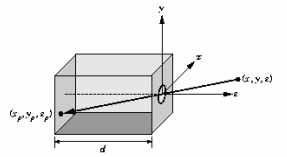
Object- Light- Viewer



Human Visual System



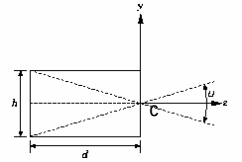
Pinhole Camera



- Small hole in the center of one side
- Film placed on opposite side
- Need long exposure

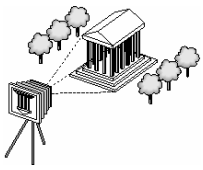
www.pinhole.com

Pinhole Camera Model



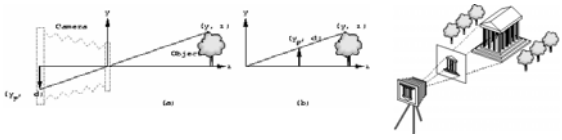
- Models geometry of image formation
- Image (projection) plane
- Optical center, C
- Focal length, d
- Projection: image of a point is a point
- Infinite Depth of Field: every point within field of view is in focus
- In basic computer graphics, all objects are in focus

Real Camera



- Creating a computer generated image is similar to forming an image using an optical system

Synthetic Camera Model: simulates pinhole camera

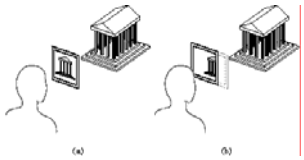


- Place optical (projection) center behind projection plane, so image is not inverted

  Center of Projection

  Projection Plane

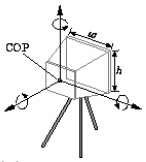
Synthetic Camera: clipping window



- Make image finite dimensions by placing Clipping Rectangle/Window in the projection plane
- The clipping rectangle acts as a window through which a viewer located at center of projection observes the world.

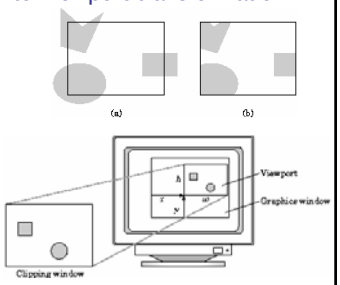
Graphics Systems

- Most API are based on the synthetic camera model
- They provide functions to specify
  - Objects (defined by sets of primitives and attributes)
  - Viewer
    - Projection type
    - Camera position
    - Camera orientation
    - Focal length (effects image size)
    - Image plane size (clipping rectangle)



2D Viewing: Window to Viewport transformation

- Viewing/clipping window: part of projected image that is being viewed. In view/projection coordinates
- Viewport: part of X-window (output window) where the viewing window is mapped. In normalized device coordinates (0 to 1). Device independent.
- Screen (device specific) coordinates, integers.



Window to viewport transformation

- Range map: given values in range A, map them linearly in range B

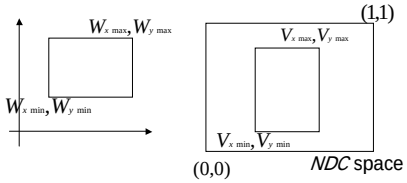
$$A = [A_{min}, A_{max}] \xrightarrow{r} B = [B_{min}, B_{max}]$$

$$A_{min} <= a <= A_{max}$$

$$b = r(a) = (B_{max} - B_{min})(a - A_{min}) / (A_{max} - A_{min}) + B_{min}$$

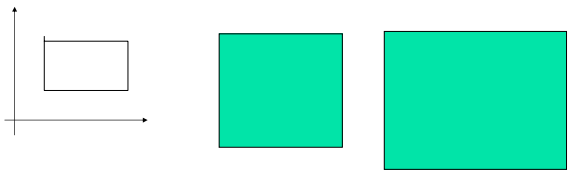
Window to viewport transformation (4.11)

- From world/view coordinates to NDC space
- Simply 2D case of range map
- Apply range map in x and y independently



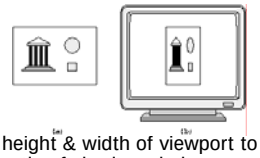
2D Viewing: window to screen coordinates (4.130)

World coordinates *Normalized Device coord.* *Screen coord. (device spec.)*



Aspect Ratio

- Aspect Ratio of a rectangle is the ratio of the rectangle's width to its height

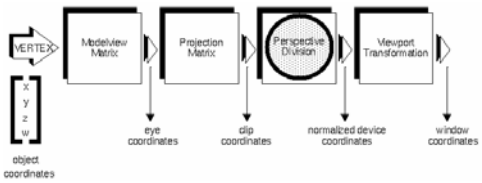


- Adjust height & width of viewport to match the aspect ratio of viewing window.

Coordinate systems used

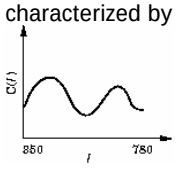
- Modeling coordinates: for specifying an object
- World coord.: for placing the object in a scene
- Viewing (eye) coord.: with respect to a coordinate system attached to the camera
- Clipping coord.: with respect to clipping window
- Normalized device coordinates: 2D, after projection, but device independent
- Physical device coordinates

Graphics Pipeline Transformations and Coordinate Systems



Color

- Light is electromagnetic radiation
- Visible light, 350 (blue) – 780 (red) nm
- Color light is characterized by a distribution $C(\lambda)$



- Monochromatic light – single frequency

Color Attributes

- Color is characterized also by attibutes
  - Hue, dominant wavelength
  - Brightness, intensity
  - Saturation, purity
  - Chromaticity, hue and saturation
- Color on CRT display is created by exciting R,G,B phosphor. This approach is based on the three color theory

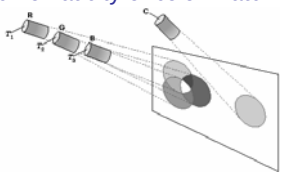
Three color Theory

- Color is described as aditive mixture of 3 primary colors, R,G,B $T_1\mathbf{R} + T_2\mathbf{G} + T_3\mathbf{B}$
- , the coefficients are called tristimulus values
- Tristimulus values – determine the intensities of the primary colors
- Three-color theory is based on popular theory, Thrichromatic theory, of human color perception

Trichromatic Theory of Human Color Perception

- Rushton:"...color vision is mediated by 3 different kinds of retinal receptors, each responding best to light from different part of the visible spectrum." Each receptor is characterized by a sensitivity curve.
- Color perception is facilitated by the cones in the retina. There 3 types of pigments in the cones.
- The trichromatic theory is based on empirically established trichromaticity of color matching

Trichromaticity of color matching



- By adjusting the radiance of the three lights of primary colors, R,G,B, match the fourth color, C.
- Match is measured by the level of excitement of the photo receptors in the cones of the retina
- The trichromatic theory predicts **if two colors will look alike** to an objective viewer, but it does not characterized how actually the colors will look to an observer.
- The theory is proven wrong by Land, 1977. Color perception is subjective. It is not related solely to the wavelength (physics) of the light.

Color in Computer Graphics

- Based on the tricolor theory
- Color is described usually by triple of numbers
- Different CG color models differ by the meaning of those triples
- There are 3 different context in which color is discussed in CG
  - Modeling/rendering
  - RGB monitor space, a triple produces here a particular color on a particular display
  - Storage and communication

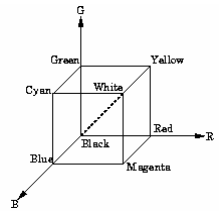
RGB Color Model

- Traditional model for CG
- R,G,B primaries
- The triple (r,g,b) gives relative weights, between 0 and 1, to the 3 primary colors in an additive system

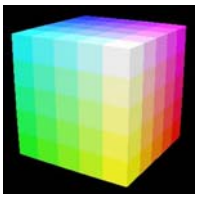
$$C = rR + gG + bB$$

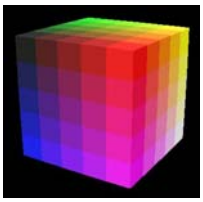
- The color gamut (all available colors) is represented by RGB color cube

Color Cube



Color Cube





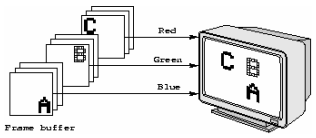
Problems with RGB Space

- Perceptually nonlinear
  - Equal distances in space do not correspond to perceptually equal sensations
  - Same color sensation may result from multiple (r,g,b) triples
  - At low RGB values a noticeable change is produces very slowly. At high values the change is produced fast.

Comments on RGB Space

- With 3 primaries only a subset of the human perception color space can be generated, and reproduced on a monitor
- (RGB) not good color descriptors for human use. There are better color models for use by people, (H,S,V).
- RGB is good enough approximation for CG
- Convenient for hardware implementations

Frame Buffer Configurations



•RBG color: FB bitplanes drive DAC directly (control guns)

glColor3f(1.0, 0.0, 0.0)

Indexed color

- Index color: FB bitplanes specify index in a lookup color table
  - N bit planes → 2^N entries in table
  - Each entry w-bits, w>=N
  - Thus, a small depth FB, N, can have more colors available, 2^w, but only 2^N simultaneously are available

Graphics Programming

Basic 2D Programs in OpenGL

Vertex

- Vertex – a location in space
- Defines the atomic geometric objec of our graphics system
- Point in space – one vertex
  Line Segment , specified by two vertices
- glVertex*

Objects

- Usually defined by a set of vertices

```
glBegin(GL_POLYGON);
   glVertex3f(0.0, 0.0, 0.0);
   glVertex3f(0.0, 1.0, 0.0);
   glVertex3f(0.0, 0.0, 1.0);
glEnd( );
```


glVertex*

- glVertex*, were * can be *nt* or *ntv*
- n signifies number of dimensions
  t denotes the data type, integer (i), float (f), double (d)
  v pointer to an array
- Ex: glVertex2f(1.0, 3.5)

Declarations

- OpenGL types
  GLfloat
  GLint
- Two dimensions example:
  glVertex2i(Glint x, Glint y)
- Three dimensions example:
  glVertex3f(Glfloat x, Glfloat y, Glfloat z)
- Using an array
  GLfloat vertex[3]
  glVertex3fv(vertex)

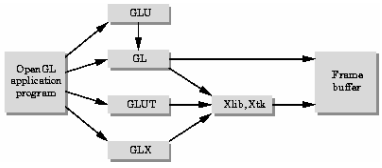
OpenGL Graphics Functions

1. Primitive – atomic or low-level objects that can be displayed (points, line segments, polygons). *What*
2. Attribute – *how* primitives appear on the screen (color, type, fill)
3. Viewing – specify our view: choose position and orientation, select lens to clip our view
4. Transformation – rotate, translate, scale an object
5. Input – deal with diverse forms of input (keybord, mouse), handled actually by GLUT not GL
6. Control – set, get system parameters

OpenGL Interface

- Function names begin with gl
  glColor3f()
- Libraries:
  GLU – Graphics Utility Library
  GLUT – GL Utility Toolkit
  GLX – X-Windows Extension
- Include Files:
  #include <GL/glut.h>

Library Organization



Hello World!

- Always your first program
- Very simple


```
/* Simple openGL program to display blue square */
#include <GL/glut.h>
void init (void);
void display(void);
int main(int argc, char** argv)
{
/* GLUT negotiates with the window system:
   Declare initial window size, position, size, display mode */
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello world!");

   init (); /* you write this function, set up camera and view */
   glutDisplayFunc(display);      /* register display callback */
   glutMainLoop();  /* waits for events, executes display func */
   return 0;   /* ANSI C requires main to return int. */ }
```



```
/* set parameters on initialization */
void init (void)
{
   /* select clearing color white */
   glClearColor (1.0, 1.0, 1.0, 1.0);
   /* set type of camera projection used, orthographic here */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
```

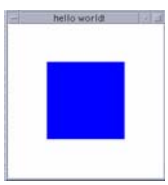


```
/* isplay function, put drawing commands here */
void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT); /* clear all pixels  */

   /* draw blue polygon (rectangle) with corners at
    * (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)     */
   glColor3f (0.0, 0.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd();

   /* don't wait! start processing buffered OpenGL routines  */
   glFlush ();
}
```


Hello World!



Homework

- Read Angel Ch 2
- Assignment 1: Edit the hello.cpp program
  - Display instead of a square, display a triangle with two vertices the bottom two vertices of the square, and a third vertex- the middle of the top side of the square
  - Due: in a week in class. Submit a hard copy of your program
  - Document your code