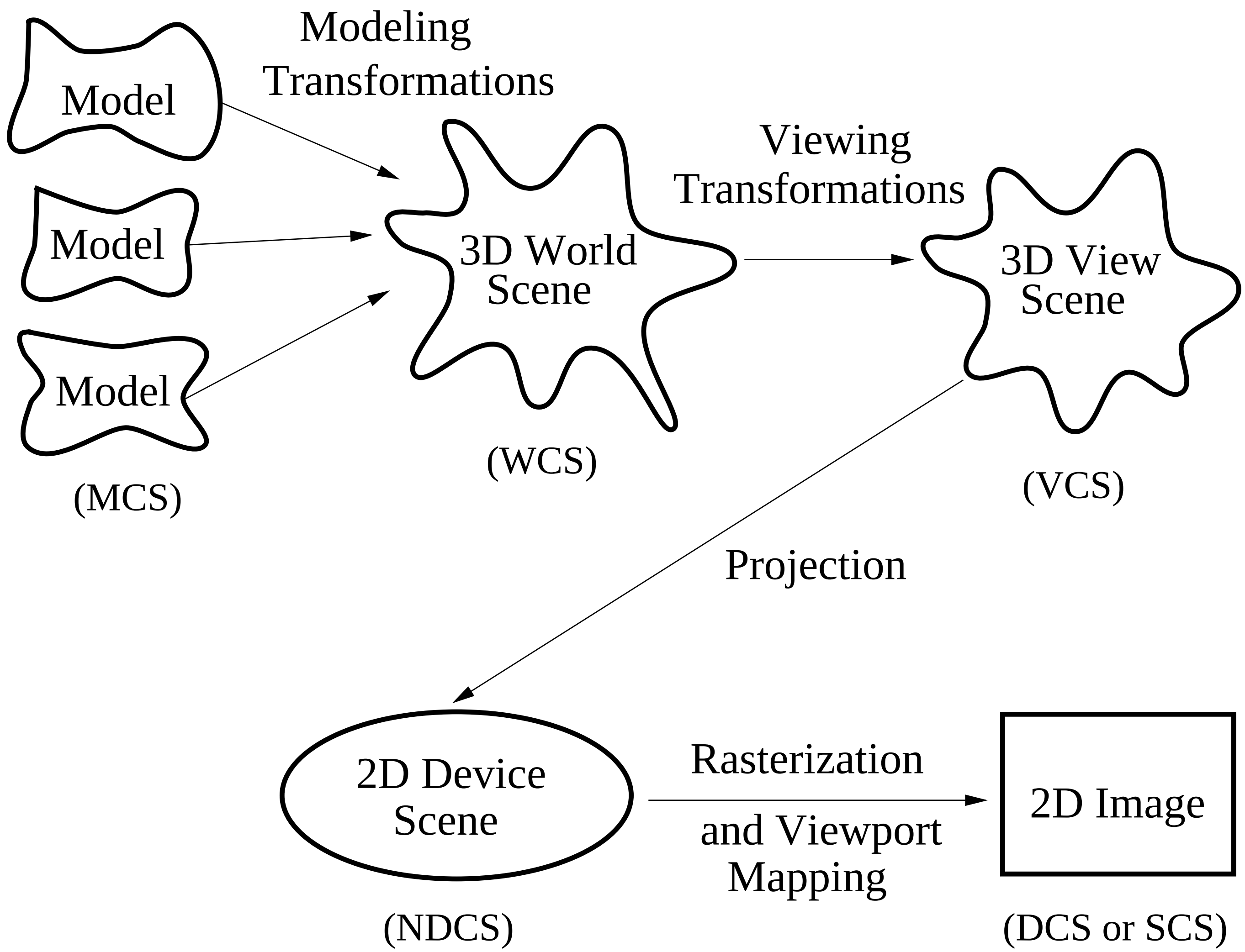
Graphics Rendering Pipeline



- Coordinate Systems
  - MCS: Modeling Coordinate System
  - WCS: World Coordinate System
  - VCS: Viewer Coordinate System
  - NDCS: Normalized Device Coordinate System
  - DCS or SCS: Device Coordinate System or, equivalently, Screen Coordinate System

  Keeping the coordinate systems straight is an important key to understanding a rendering system.

- Pipeline stages: Transform -¿ Clip -¿ Project -¿ Rasterize
  - Convert primitives in the MCS to primitives in the WCS.
  - Add derived information: shading, texture, shadows.
  - Remove invisible primitives as convertion to VCS.
  - Project primitives from VCS to NDCS
  - Convert primitives into the DCS (from NDCS) to *pixels* in a *raster image*.
- Transformations: Coordinate system conversions can be represented with matrix-vector multiplications. Matrices are of size 4x4 for 3D graphics

Rendering Primitives

Models are typically composed of a large number of *geometric primitives*. The *only* rendering primitives typically supported in hardware are

- Points (single pixels)
- Line segments
- Polygons (usually restricted to *convex polygons*).

Modeling primitives include these, but also

- Piecewise polynomial (spline) curves
- Piecewise polynomial (spline) surfaces
- Implicit surfaces (quadrics, blobbies, etc)
- Other...

A software renderer may support these modeling primitives directly, or they may be converted into polygonal or linear approximations for hardware rendering.

Algorithms

A number of basic algorithms are needed:

- Transformation: convert representations of primitives from one coordinate system to another.
- Clipping/Hidden Surface Removal: Remove primitives and parts of primitives that are not visible on the display.
- Rasterization: Convert a projected screen-space primitive to a set of pixels.

Later, we will look at some more advanced algorithms:

- Picking: Select a 3D object by clicking an input device over a pixel location.
- Shading and Illumination: Simulate the interaction of light with a scene.
- Texturing and Environment Mapping: Enhancing the realism
- Animation: simulate movement by rendering a sequence of frames.

Application Programming Interfaces

- Application Programming Interfaces (APIs) provide access to rendering hardware:
  - Xlib: 2D rasterization.
  - PostScript: 2D transformations, 2D rasterization
  - GL, OpenGL: 3D pipeline

- APIs hide which parts of the rendering are actually implemented in hardware by simulating the missing pieces in software, usually at a loss in performance.
- For 3D interactive applications, we might modify the scene or a model directly or just the viewing information.
- After each modification, usually the images needs to be regenerated.
- We need to consider how to interface to input devices in an asynchronous and device independent fashion. APIs have also been defined for this task; we will be using X11 through Glut

Device Independence

In this module, we

- Consider display devices for computer graphics:
  - calligraphic devices
  - raster devices
  - CRTs
  - direct vs. pseudocolor frame buffers
- Discuss the problem of device independence:
  - window-to-viewport mapping
  - normalized device coordinates

Calligraphic and Raster Devices

- Calligraphic display devices draw polygon and line segments directly:
  - plotters
  - direct beam control CRTs
  - laser light projection systems
- Raster display devices represent an image as a regular grid of *samples*.
  - Each sample is usually called a *pixel* or, less commonly, a *pel*.
  - Both are short for *picture element*.
  - Rendering requires *rasterization algorithms* to quickly determine a sampled representation of geometric primitives.

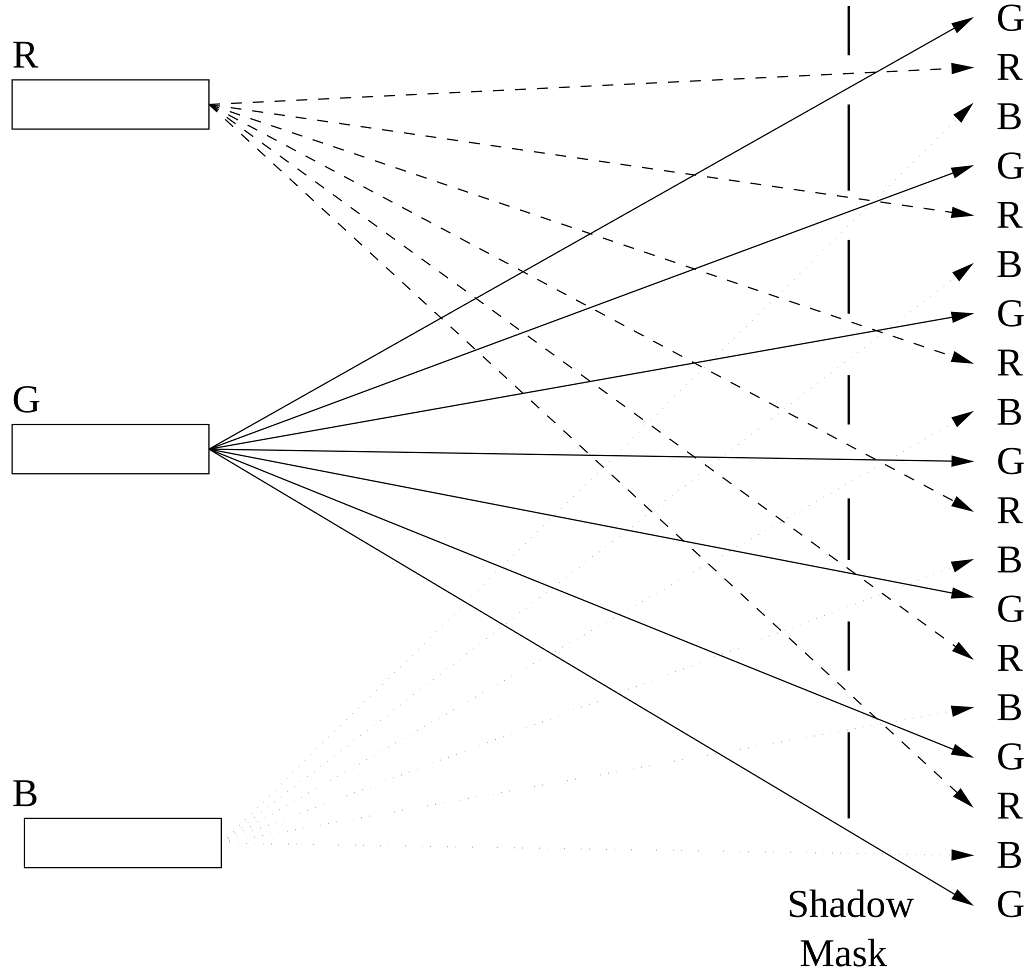
How a Monitor Works

- Raster Cathode Ray Tubes (CRTs) are the most common display device today.
  - capable of high resolution
  - good color fidelity
  - high contrast (100:1)
  - high update rates

  An electron beam is continually scanned in a regular pattern of horizontal *scanlines*.
- *Raster images* are stored in a *frame buffer*.
- *Frame buffers* are composed of *VRAM* (video RAM).
- VRAM is dual-ported memory capable of
  - Random access.
  - Simultaneous high-speed serial output: A built-in *serial shift register* can output an entire scanline at a high rate synchronized to a *pixel clock*.

  At each pixel location in a scanline, the intensity of the electron beam is modified by the pixel value being shifted synchronously out of the VRAM.

- Color CRTs have three different colors of phosphor and three independent electron guns.
- *Shadow masks* only allow each gun to irradiate one color of phosphor.



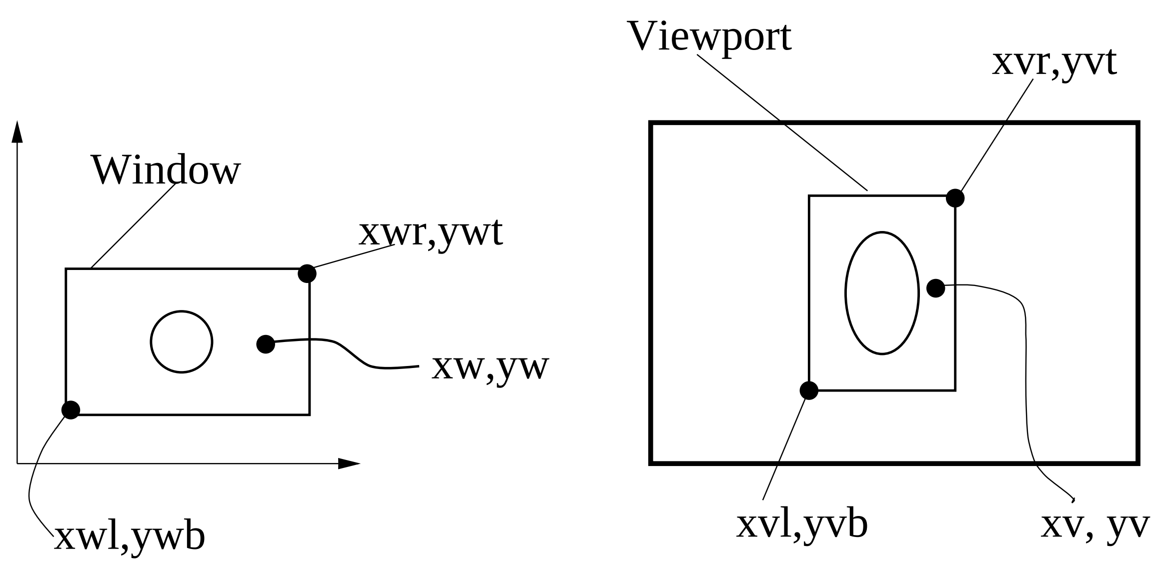
- Color is specified either

- directly, using three independent intensity channels, or
- indirectly, using a *color lookup table* (LUT).

  In the latter case, a *color index* is stored in the frame buffer.

Window to Viewport Mapping

- Start with 3D scene, but eventually project to 2D scene.
- 2D scene is infinite plane. Device has a finite visible rectangle. What do we do?
- Answer: map rectangular region of 2D device scene to device.
  - Window: rectangular region of interest in scene.
  - Viewport: rectangular region on device.
  - Usually, both rectangles are aligned with the coordinate axes.



- Window point (x_w, Y_w) maps to viewport point (x_v, y_v).
  - Window has corners (x_{wl}, y_{wb}) and (x_{wr}, y_{wt});
    Viewport has corners (x_{vl}, y_{vb}) and (x_{vr}, y_{vt});
  - Length and height of the window are L_w and H_w
    Length and height of the viewport are L_v and H_v
- Proportionally map each of the coordinates according to:

$$\frac{\Delta x_w}{L_w} = \frac{\Delta x_v}{L_v}$$

$$\frac{\Delta y_w}{H_w} = \frac{\Delta y_v}{H_v}$$

- To map x_w to x_v:

$$\frac{x_w - x_{wl}}{L_w} = \frac{x_v - x_{vl}}{L_v}$$

$$\Rightarrow x_v = \frac{L_v}{L_w}(x_w - x_{wl}) + x_{vl}$$

and similarly for y_v.

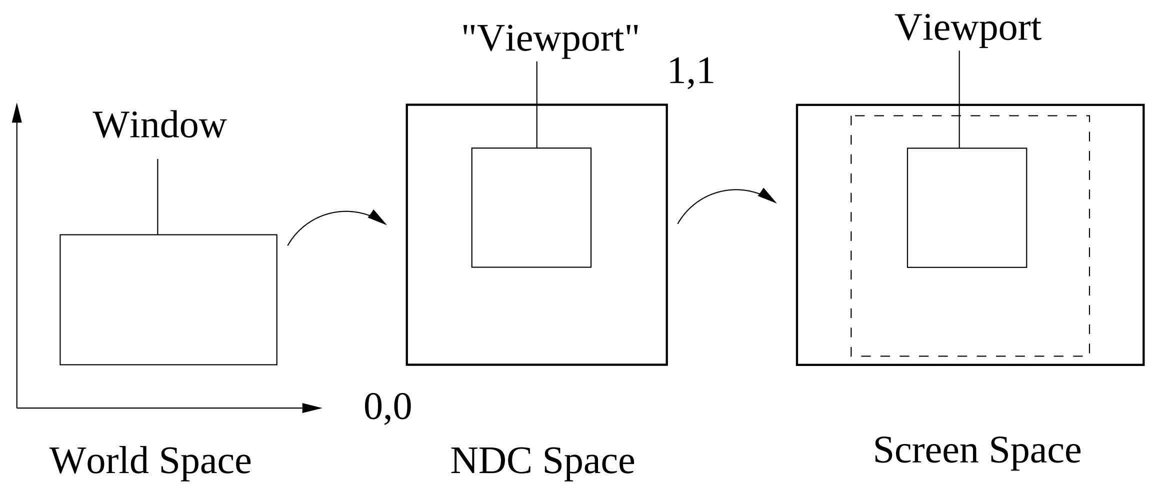
- If $H_w/L_w \neq H_v/L_v$ the image will be distorted.
  These quantities are called the *aspect ratios* of the window and viewport.

Normalized Device Coordinates

- Where do we specify our viewport?
- Could specify it in device coordinates, BUT, suppose we want to run our program on several different hardware platforms or on different graphic devices.
- Two common conventions for DCS:
  - Origin in the lower left corner, with x to the right and y upward.
  - Origin in the top left corner, with x to the right and y downward.
- Many different resolutions for graphics display devices:
  - Workstations commonly have 1280×1024 frame buffers.
  - A PostScript page is 612×792 points, but 2550×3300 pixels at 300dpi.
  - And so on...
- Aspect ratios may vary...

- If we map directly from WCS to a DCS, then changing our device requires rewriting this mapping (among other changes).
- Instead, use Normalized Device Coordinates (NDC) as an intermediate coordinate system that gets mapped to the device layer.
- Will consider using only a square portion of the device.

  Windows in WCS will be mapped to viewports that are specified within a unit square in NDC space.
- Map viewports from NDC coordinates to the screen.



Reading Assignment

Chapter 1 pages 1 - 36, of Recommended Text

(Recommended Text: Interactive Computer Graphics, by Edward Angel, 4th edition, Addison-Wesley)

Please track the News section of the Course Web Pages for the most recent Announcements related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics24/cs354/)