Hardware rendering
Graphics hardware

- Intel, AMD, Mac, IBM, HP (GHz)
- NVIDIA, ATI
- Screen

- CPU
- RAM
- GPU
- Frame buffer

- 4 GB (32 bit)
- 100’s GB (64 bit)
- PCI, AGP2x-8x, PCIe
- 256 MB, 2GB, ...
- 0.01-9 Mpixels
Fixed pipeline hardware support

• What hardware does
  – Transformation
  – Projection
  – Clipping
  – Lighting (ambient, diffuse, specular)
  – Shading (Gouraud, Phong)
  – Texture, environment, reflection, perspective, and shadow mapping
  – Antialiasing
  – Some image processing

• and does not do
  – Model the scene geometry for you
  – Light the scene for you
  – Find nice paths in the model
  – Animate the model for you
  – Implement the laws of physics
  – Create and manage windows
  – Decide which parts of the model to process for each frame
Programmable pipeline

- **Vertex programs**
  - Define your own transformation
  - Define your own lighting
  - Compute per vertex values needed at pixel level

- **Pixel programs**
  - aka pixel shaders
  - aka fragment programs
  - Define your own shader (way to compute color from vertex data)

- **Geometry programs**
  - Programmability at primitive level
  - Define your own primitive
Input to graphics hardware

- **CPU**
- **RAM**

- **GPU**
  - Frame buffer

- **Screen**

**Flow:**
- **CPU** sends data to **GPU**
- **GPU** processes data and sends it to **Screen**

**Data in RAM:**
- Clear buffer
- Desired view
- Geometry
- Lights

**Data in special language:**
- Sent to HW in special language define by graphics API (e.g. OpenGL, DirectX)
- Ensures independence between application and hardware details
- The HW driver takes API calls and converts them to low level HW specific calls
OpenGL: clear buffers

• void TiffView::GLFrameSetup() {
  
  
  
  
  • glClearColor(0.0f, 0.0f, 0.5f, 1.0f);
  • glClearStencil(0);

  • glClear( GL_COLOR_BUFFER_BIT |
             GL_DEPTH_BUFFER_BIT |
             GL_STENCIL_BUFFER_BIT);

  
  
  }
Specify desired view

• Convert “software” planar pinhole camera (PHC) model to hardware view
  – Step 1: specify intrinsics
    • Analogous to PHC constructor
    • At the beginning of session & when intrinsics change (e.g. after change of FOV, and of resolution operations)
    • Many ways of doing it
  – Step 2: specify extrinsics (camera location and orientation)
    • Every time the view changes (for every frame)
    • Many ways of doing it
PHC to OpenGL: intrinsics
PHC to OpenGL: intrinsics

void TiffView::InitializeGL(PHCamera *phc, float hither, float yon) {
    int dvW = phc->w, dvH = phc->h;
    glViewport(0, 0, dvW, dvH);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    float wf = phc->b.Length() * dvW;
    float hf = phc->b.Length() * dvH;
    // specify rectangle on hither plane and distances to hither and yon planes
    glFrustum (-wf/2.0f*scalef, wf/2.0f*scalef, -hf/2.0f*scalef, hf/2.0f*scalef, hither, yon);
    glMatrixMode (GL_MODELVIEW);  // default matrix mode
}
PHC to OpenGL: extrinsics

- void TiffView::SetGLView(PHCamera *phc) {
  - Vex3 eye, look, down;
  - eye = phc->GetC();
  - look = phc->GetLookAtPoint();
  - down = (phc->Getb()).UnitVector();
  - int dvW = phc->w, dvH = phc->h;
  - glLoadIdentity();
  - // COP, point to look at (C + axb), up vector
  - gluLookAt(eye[0], eye[1], eye[2], look[0], look[1], look[2], -down[0], -down[1], -down[2]);
  - }


void TriangleMesh::RenderSharedVertexHW(int renderMode) {
    if (renderMode & TRIANGLE_RENDERMODE_WF) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        if (glColor[0] != -1.0f)
            glColor4fv(glColor);
    } else
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glEnableClientState(GL_VERTEX_ARRAY);
    // similar for array of colors and array of texture coordinates
    if (glNormals)
        glEnableClientState(GL_NORMAL_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, glVertices);
    if (glNormals)
        glNormalPointer(GL_FLOAT, 0, glNormals);
    glDrawElements(GL_TRIANGLES, 3*trianglesN, GL_UNSIGNED_INT, connectivity);
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisableClientState(GL_VERTEX_ARRAY);
}