Chapter 2

Basics of Computer Graphics: Rendering Pipeline/OpenGL

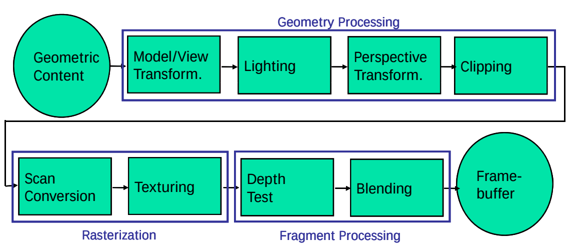
Rendering Tasks (no particular order)

- Project 3D geometry onto image plane
  - Geometric transformations
- Determine which primitives/parts of primitives are visible
  - Hidden surface removal
- Determine which pixels geometric primitive covers
  - Scan conversion
- Compute color of every visible surface point
  - Lighting, shading, texture mapping

Todo's reminder

- Discussion Group: register
- Assignment 1
  - Test programming environment on lab computers/Set up programming environment on your laptop (optional)
  - Start assignment (have most background after class today)
- Reading (in Shirley: Introduction to CG)
  - Math refresher: Chapters 2, 4
    - Lots of math coming soon– be ready !!!
  - Background on graphics: Chapter 1

The Rendering Pipeline



Rendering

Goal:
- Transform (3D) computer models into images
- Photo-realistic (or not)

Interactive rendering:
- Fast, but until recently low quality
- Roughly follows a fixed patterns of operations
  - **Rendering Pipeline**

Offline rendering:
- Ray-tracing
- Global illumination

Rendering Pipeline

- Abstract model of
  - sequence of operations to transform geometric model into digital image
  - graphics hardware workflow
- Underlying API (application programming interface) model for programming graphics hardware
  - OpenGL
  - Direct 3D
- **Actual implementations vary**

Discussion

- Advantages of pipeline structure
  - Logical separation of different components, modularity
  - Easy to parallelize:
    - Earlier stages can already work on new data while later stages still work with previous data
    - Similar to pipelining in modern CPUs
    - But much more aggressive parallelization possible (special purpose hardware!)
    - Important for hardware implementations!
  - Only local knowledge of the scene is necessary

Rendering Pipeline Implementation: OpenGL/Glut

Discussion

- Disadvantages:
  - Limited flexibility
  - Some algorithms would require different ordering of pipeline stages
    - Hard to achieve while still preserving compatibility
  - Only local knowledge of scene is available – doesn't support:
    - Shadows
    - Global illumination

OpenGL

- API for graphics hardware
- Started in 1989 by Kurt Akeley
- Designed to exploit GPU
- Implemented on many different platforms
- Low level, powerful flexible
- Pipeline processing
  - Communication via state setting
  - Event driven

(Tentative) Lecture Syllabus

- Introduction + Rendering Pipeline (week 1/2)
- Transformations (week 2/3)
- Scan Conversion (week 4/5)
- Clipping (week 5)
- Hidden Surface Removal (week 6/7)
- Review & Midterm (week 7)
  - Midterm: Oct 20
- Lighting Models (week 8)
- Texture mapping (week 9/10)
- Review & Midterm (week 10)
  - Midterm: Nov 10
- Ray Tracing (week 11)
- Shadows (week 11/12)
- Modeling (content creation) (week 12/13)
- Review (last lecture)

Graphics State (global variables)

- Set state once, remains until overwritten
  - glColor3f(1.0, 1.0, 0.0) → set color to yellow
  - glSetClearColor(0.0, 0.0, 0.2) → dark blue bg
  - glEnable(LIGHT0) → turn on light
  - glEnable(GL_DEPTH_TEST) → hidden surf.

GLUT: OpenGL Utility Toolkit

- **Event driven !!!**

```
int main(int argc, char **argv)
{
      glutInit( &argc, argv );
      glutInitDisplayMode( GLUT_RGB |
                           GLUT_DOUBLE | GLUT_DEPTH);
      glutInitWindowSize( 640, 480 );
      glutCreateWindow( "openGLDemo" );
      glutDisplayFunc( DrawWorld );
      glutIdleFunc(Idle);
      glClearColor( 1,1,1 );
      glutMainLoop();

      return 0;      // never reached
}
```

GLUT Example

```
void Idle() {
      angle += 0.05;
      glutPostRedisplay();
}
```

Event-Driven Programming

- Main loop not under your control
  - vs. procedural
- Control flow through event callbacks
  - redraw the window now
  - key was pressed
  - mouse moved
- Callback functions called from main loop **when events occur**
  - mouse/keyboard, redrawing...

GLUT Input Events

```
  // you supply these kind of functions
void reshape(int w, int h);
void keyboard(unsigned char key, int x, int y);
void mouse(int but, int state, int x, int y);

  // register them with glut
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse);
```

OpenGL/GLUT Example

```
void DrawWorld() {
      glMatrixMode( GL_PROJECTION );
      glLoadIdentity();
      glMatrixMode( GL_MODELVIEW );
      glLoadIdentity();
      glClear( GL_COLOR_BUFFER_BIT );
      angle += 0.05;
      glRotatef(angle,0,0,1);
      ...  // draw triangle
      glutSwapBuffers();
}
```

GLUT and GLU primitives

```
gluSphere(...)
gluCylinder(...)
glutSolidSphere(GLdouble radius, GLint slices, GLint stacks)
glutWireSphere(...)
glutSolidCube(GLdouble size)
glutWireCube(...)
glutSolidTorus(...)
glutWireTorus(...)
glutSolidTeapot(...)
glotWireTeapot(...)
```

- Note:
  - Have limited set of parameters
  - Control via global transformations (see a1 template)
  - **Need to save/restore setting**

GLUT and GLU primitives

- Example (from a1): draw ball at random positions

```
// Setup transformation matrix for this ball.

glPushMatrix();   → Save previous state

glTranslatef(x_, y_, 0);
// Now have coordinate system with 0,0,0 at ball centre
// DRAW IT!
glutSolidSphere(radius_, 12, 4);
// Restore the transformation matrix.

glPopMatrix();   → Restore previous state
```

Geometry Pipeline

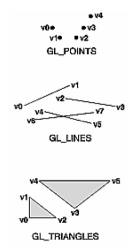
- How to interpret geometry
  - glBegin(<*mode of geometric primitives*>)
  - *mode* = GL_TRIANGLE, GL_POLYGON, etc.
- Feed vertices
  - glVertex3f(-1.0, 0.0, -1.0)
  - glVertex3f(1.0, 0.0, -1.0)
  - glVertex3f(0.0, 1.0, -1.0)
- Done
  - glEnd()

GLUT and GLU primitives

- Basic Transformations:

```
// Different basic transformations
glTranslatef(…);
glRotatef(…);
glScalef(…);
```

Open GL: (Some) Primitives



glPointSize(float size);
glLineWidth(float width);
glColor3f(float r, float g, float b);
....

- TRIANGLE…

```
glColor3f(0,1,0);
glBegin( GL_TRIANGLES );
  glVertex3f( 0.0f, 0.5f, 0.0f );
  glVertex3f( -0.5f, -0.5f, 0.0f );
  glVertex3f( 0.5f, -0.5f, 0.0f );
glEnd();
```

Depth buffer

- For visibility (hidden surface removal)
  - stores a z-value for every pixel
  - smaller z means "closer"

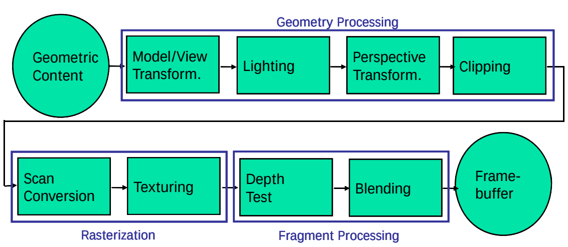
```
  // allocate depth buffer
glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

  // enabling the depth test
glEnable( GL_DEPTH_TEST );

  // clearing the depth buffer for each frame
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Assignment 1

- Experience OpenGL & GLUT
- Have FUN
- Description:
  http://www.ugrad.cs.ubc.ca/~cs314/Vsep2011/a1/a1.pdf
- Deadline: Sep 23

Rendering Pipeline in (More) Detail

The Rendering Pipeline



(Not only) Geometric Content

- Needs to represent models for
  - Geometric primitives
  - Relations between different primitives (transformations)
  - Object materials
  - Light sources
  - Camera

Geometric Primitives

- Different philosophies:
  - Collections of complex shapes
    - Spheres, cones, cylinders, tori, …
  - One simple type of geometric primitive
    - Triangles or triangle meshes
  - Small set of complex primitives with adjustable parameters
    - E.g. “all polynomials of degree 2”
    - Splines, NURBS (details in CPSC 424)
    - Implicits

Geometric Primitives

- Mathematical representations:
  - Explicit functions
  - Parametric functions
  - Implicit functions

Explicit Functions

- Curves:
  - y is a function of x: $y := \sin(x)$
  - Only works in 2D
- Surfaces:
  - z is a function of x and y: $z := \sin(x) + \cos(y)$
  - Cannot define arbitrary shapes in 3D

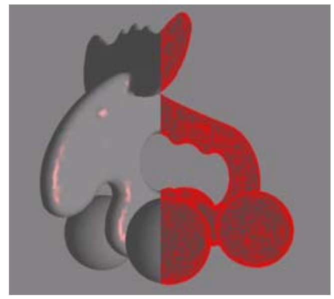
Parametric Functions

- Curves:
  - 2D: x and y are functions of a parameter value t
  - 3D: x, y, and z are functions of a parameter value t

$$C(t) := \begin{pmatrix} \cos(t) \\ \sin(t) \\ t \end{pmatrix}$$

Geometric Content

- Triangles and Triangle Meshes:

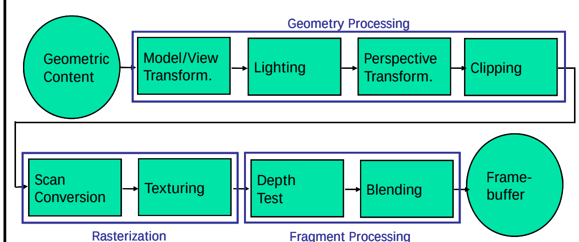


Parametric Functions

- Surfaces:
  - Surface S is defined as a function of parameter values s, t
  - Names of parameters can be different to match intuition:

$$S(\phi,\theta) := \begin{pmatrix} \cos(\phi)\cos(\theta) \\ \sin(\phi)\cos(\theta) \\ \sin(\theta) \end{pmatrix}$$

The Rendering Pipeline



Geometric Content

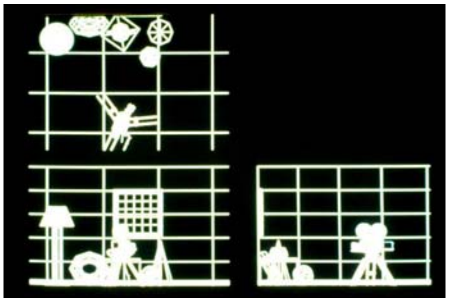
- Implicit Surfaces:
  - Surface defined by zero set (roots) of function
  - E.g:

$$S(x,y,z): x^2 + y^2 + z^2 - 1 = 0$$

Modeling and Viewing Transformations

- Placing objects - Modeling transformations
  - Map points from object coordinate system to world coordinate system
- Placing camera - Viewing transformation
  - Map points from world coordinate system to camera (or eye) coordinate system

Modeling Transformations: Object Placement

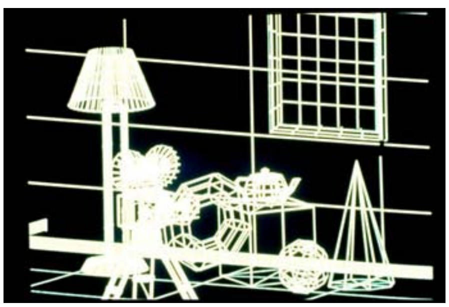


Modeling & Viewing Transformation

- Linear transformations
  - Rotations, scaling, shearing
  - Can be expressed as 3x3 matrix
  - E.g. rotation:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Viewing Transformation: Camera Placement



Modeling & Viewing Transformation

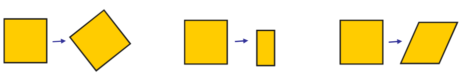
- Affine transformations
  - Linear transformations + translations
  - Can be expressed as 3x3 matrix + 3 vector
  - E.g. rotation + translation:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

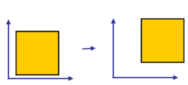
  - Another representation: 4x4 homogeneous matrix

Modeling & Viewing Transformations

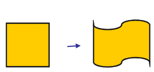
- Types of transformations:
  - Rotations, scaling, shearing



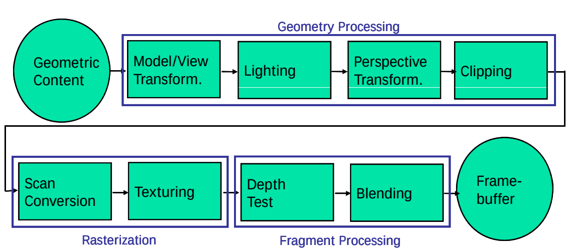
  - Translations



  - Other transformations (not handled by rendering pipeline):
    - Freeform deformation



The Rendering Pipeline



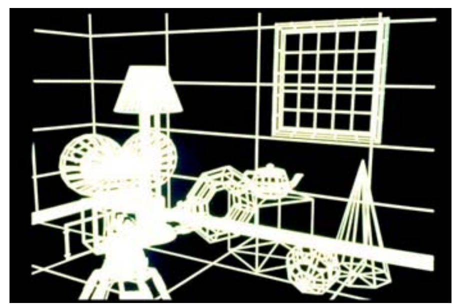
Lighting

Perspective Transformation

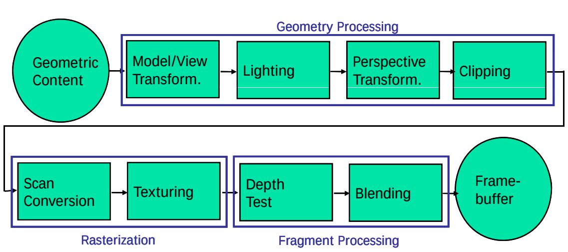
- Purpose:
  - Project 3D geometry to 2D image plane
  - Simulates a camera
- Camera model:
  - Pinhole camera (single view point)
  - Other, more complex camera models also exist in computer graphics, but are less common
    - Thin lens cameras
    - Full simulation of lens geometry

Complex Lighting and Shading

Perspective Projection

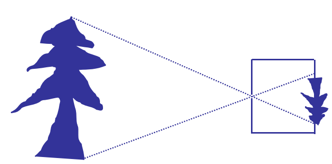


The Rendering Pipeline



Perspective Transformation

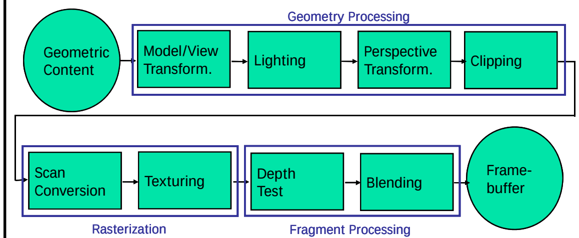
- Pinhole Camera:
  - Light shining through a tiny hole into a dark room yields upside-down image on wall



Perspective Transformation

- Pinhole Camera

The Rendering Pipeline



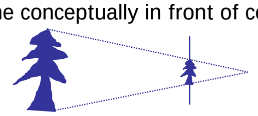
Pinhole Camera - Camera Obscura

Clipping

- Removing invisible geometry
  - Geometry outside viewing frustum
  - Plus too far or too near one

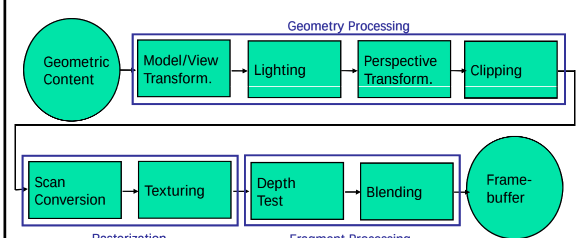
Perspective Transformation

- In computer graphics:
  - Image plane conceptually in front of center of projection



  - Perspective transformations – subset of projective transformations
  - Linear & affine transformations also belong to this class
  - All projective transformations can be expressed as 4x4 matrix operations

The Rendering Pipeline



Scan Conversion/Rasterization

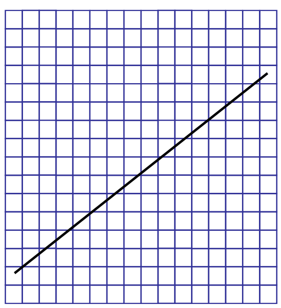
- Convert continuous 2D geometry to discrete
- Raster display – discrete grid of elements
- Terminology
  - **Pixel:** basic element on device

  - **Resolution:** number of rows & columns in device
    - Measured in
      - Absolute values (1K x 1K)
      - Density values (300 dots per inch)
  - **Screen Space:** Discrete 2D Cartesian coordinate system of the screen pixels

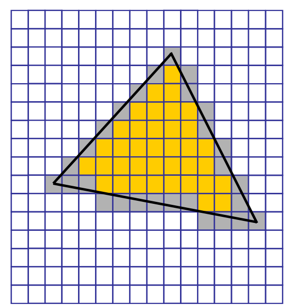
Scan Conversion

- Problem:
  - Line is infinitely thin, but image has finite resolution
  - Results in steps rather than a smooth line
    - Jaggies
    - Aliasing
  - One of the fundamental problems in computer graphics

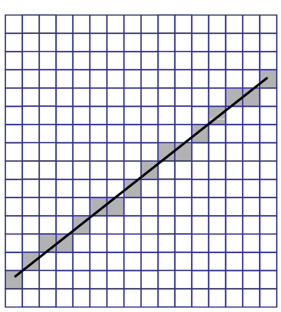
Scan Conversion



Scan Conversion

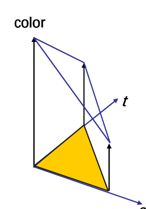


Scan Conversion



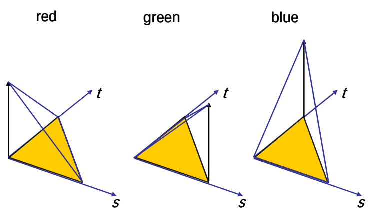
Scan Conversion

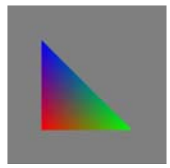
- Color interpolation
  - Linearly interpolate per-pixel color from vertex color values
  - Treat every channel of RGB color separately



Scan Conversion

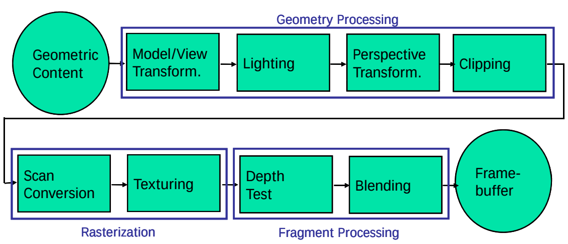
- Color interpolation
  - Example:





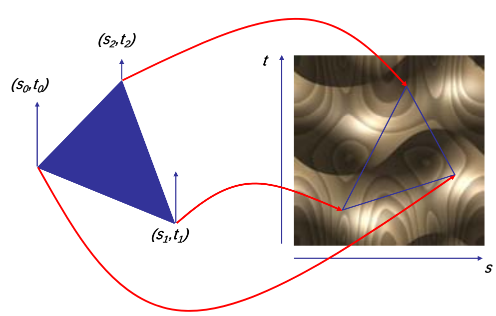
Texture Mapping

The Rendering Pipeline



Displacement Mapping

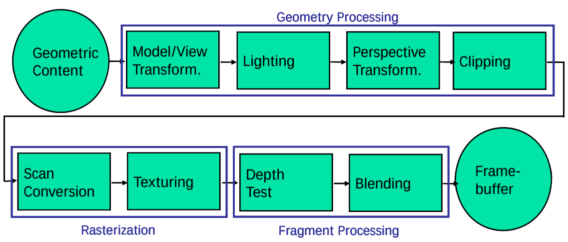
Texturing



Texturing

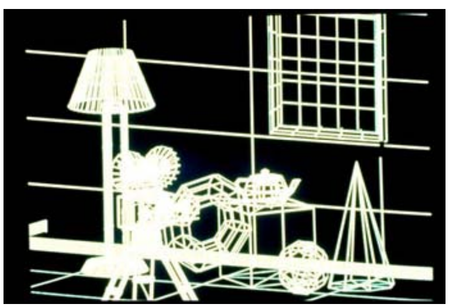
- Issues:
  - How to map pixel from texture (texels) to screen pixels
    - Texture can appear widely distorted in rendering
    - Magnification / minification of textures
  - Filtering of textures
  - Preventing aliasing (anti-aliasing)

The Rendering Pipeline



Hidden Surface Removal

Without Hidden Line Removal

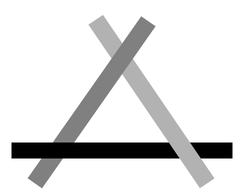


Depth Test /Hidden Surface Removal

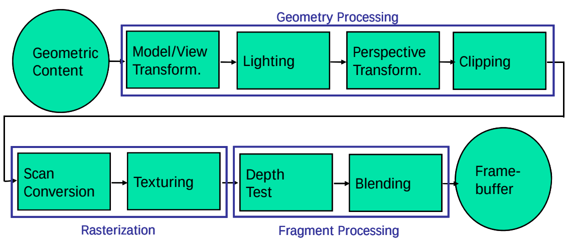
- Remove invisible geometry
  - Parts that are hidden behind other geometry
- Possible Implementations:
  - Per-fragment decision
    - Depth buffer
  - Object space decision
    - Clipping polygons against each other
    - Sorting polygons by distance from camera

Hidden Line Removal

Depth Test /Hidden Surface Removal



The Rendering Pipeline



Blending

- Blending:
  - Final image: write fragments to pixels
  - Draw from farthest to nearest
  - No blending – replace previous color
  - Blending: combine new & old values with some arithmetic operations
- Frame Buffer : video memory on graphics board that holds resulting image & used to display it

Not Handled: Reflection/Shadows