Geometric Objects and Transformations
Graphics Pipeline

• How can we render a triangle?

• High-level description:
  – Represent the 3D geometry
  – Transform it to a 2D image

• Our 3D objects are given as polygon meshes
Geometric Objects

- Which mathematical entities do we need?
  - Scalars
  - Points
  - Vectors
Geometric Objects – Scalars

• We will use the real numbers, i.e., \( \alpha, \beta \in \mathbb{R} \).

• Two (commutative and associative) operations are defined:
  – Addition: \( \gamma = \alpha + \beta \)
  – Multiplication: \( \gamma = \alpha \cdot \beta \)

• We also have identity elements, 0 and 1.
Geometric Objects – Points

• A location in space

• A point has neither size nor a shape

• Connect points with directed line segments, i.e., it make sense to write $Q = P + v$. 
Geometric Objects – Vectors

• Vectors have direction and magnitude

• What relations can we define?

  – Vectors addition:
    \[ u + v = (u_1 + v_1, u_2 + v_2, u_3 + v_3) \]

  – Scalar-Vector multiplication:
    \[ \alpha v = (\alpha v_1, \alpha v_2, \alpha v_3) \]
Geometric Objects – Vectors

• What relations can we define?
  – **Inner (dot) product:**
    \[ u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3. \]

• Commutative
  \[ u \cdot v = v \cdot u \]

• Linear
  \[ (\alpha u + v) \cdot w = \alpha u \cdot w + v \cdot w \]
Geometric Objects – Vectors

• What relations can we define?
  – **Inner (dot) product:**
    \[ u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3. \]

• Vectors are **orthogonal** if \( u \cdot v = 0 \)

• \( u, v \) are **orthonormal** if \( u \cdot v = 0 \) and \( ||u|| = ||v|| = 1 \)

• **Norm** of a vector: \( ||u|| = \sqrt{u \cdot u} \)
Geometric Objects – Vectors

• What relations can we define?
  – Inner (dot) product:
    \[ u \cdot v = \| u \| \| v \| \cos \theta. \]

• Vectors are orthogonal if \( u \cdot v = 0 \)

• \( u, v \) are orthonormal if \( u \cdot v = 0 \) and \( \| u \| = \| v \| = 1 \)

• Norm of a vector: \( \| u \| = \sqrt{u \cdot u} \)
Geometric Objects – Vectors

• What relations can we define?

  – Cross product:

    \[ u \times v = \begin{vmatrix} i & j & k \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix} \]

  • Anticommutative

    \[ u \times v = -v \times u \]

  • Linear

    \[ (\alpha u + v) \times w = \alpha u \times w + v \times w \]
Geometric Objects – Vectors

• What relations can we define?
  – Cross product:

\[
\mathbf{u} \times \mathbf{v} = \begin{vmatrix} i & j & k \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix}
\]

• \( \mathbf{u} \times \mathbf{v} \) is orthogonal to \( \mathbf{u} \) and \( \mathbf{v} \), i.e.,
  \[
  (\mathbf{u} \times \mathbf{v}) \cdot \mathbf{u} = 0 \quad \land \quad (\mathbf{u} \times \mathbf{v}) \cdot \mathbf{v} = 0
  \]

• Area of the parallelogram defined by \( \mathbf{u} \) and \( \mathbf{v} \), i.e.,
  \[
  \|\mathbf{u} \times \mathbf{v}\| = \|\mathbf{u}\|\|\mathbf{v}\| \sin \theta
  \]
Geometric Objects – Vectors

• Why do we need inner/cross products?

• Example: Back face culling

  – For each face, find its normal direction $n = u \times v$

  – Check the sign of its inner product with the viewing direction, i.e., $n \cdot e$
Transformations

• Which transformations do we need?
  – Transform (e.g., rotate) a particular object
  – Transform all of the objects together
  – Transform the camera (position or viewing)
Transformations

• In fact, all of the following frames are needed:

1. Object (model) frame
2. World frame
3. Camera (eye) frame
4. Clip frame
5. Normalized device frame
6. Window frame
7. Screen frame
Transformations

• In fact, all of the following frames are needed:

1. Object (model) frame
2. World frame
3. Camera (eye) frame
4. Clip frame
5. Normalized device frame
6. Window frame
7. Screen frame
Coordinate Systems vs. Frames

• A set of basis vectors, $u, v, w$, define a coordinate system

• Any vector can be represented in this basis
  \[ x = \alpha u + \beta v + \gamma w, \quad \alpha, \beta, \gamma \in \mathbb{R} \]

• Matrix form:
  \[ x = \begin{pmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \]
Coordinate Systems vs. Frames

• However, points cannot be represented...

• A frame is defined by \((u, v, w, P_0)\), where \(P_0\) is the origin.

• In practice, we work with \(4 \times 4\) matrices which represent the frames of our program
Frames

• How do we move objects between frames?

• Represent the object in the new frame

  – If $M$ is the change of frames matrix from the original frame to $(u, v, w, P)$

  – Apply $M^T$ to the object, i.e.,

    $$
    \begin{pmatrix}
    u_1 v_1 & w_1 & P_1 \\
    u_2 & v_2 & w_2 P_2 \\
    u_3 & v_3 & w_3 P_3 \\
    0 & 0 & 0 & 1
    \end{pmatrix}
    $$
Changing Frames

- We store all object transformations in Model and World frames in $T_m$ and $T_w$, respectively.

- Similarly, we have the Camera transformation matrix, $T_c$.

- Thus, to move from Model frame to Camera frame, we apply $T = T_c^{-1} \circ T_w \circ T_m$. Why?
Changing Frames

• $T$ positions the camera relative to the objects

• Choose: transform the objects or the camera

• The matrix $T$ is called the Model-View matrix

• Do we really need to compute inverses?

\[ (T_1 \circ T_2)^{-1} = T_2^{-1} \circ T_1^{-1} \]
Question

• How can we implement a “look at” feature?

• The camera should snap to a certain object

• Basically, a change of frames for the camera...
Visual Studio Project

• A C++ project with the following classes:

  – **Model** class: geometry, frame, ...
  – **Camera** class: frame, projection, ...
  – **Scene** class: stores objects, cameras, lights, ...
  – **Renderer** class: renders the scene to an output
  – Math related classes
Visual Studio Project – Model Class

class Model {
protected:
    virtual ~Model() {} 
    void virtual draw()=0;
    Geometry T;
    mat4 mTransform;
};
Visual Studio Project – Camera Class

class Camera {
   // constructors
   mat4 cTransform;
   mat4 projection;

public:
   void setTransformation(const mat4& T);
   void setProjection(const mat4& T);
   void LookAt(...);
   void Ortho(...);
   void Perspective(...);
   ...
};
Visual Studio Project – Scene Class

class Scene {
    vector<Model*> models;
    vector<Camera*> cameras;
    renderer *m_renderer;

public:
    Scene(Renderer *renderer);
    void AddModel(Model* model);
    void AddCamera(Model* model);
    Model* GetModel(int model_id);
    ...
    void draw();
};
Visual Studio Project – Renderer Class

class Renderer{
    float *m_outBuffer; // 3*width*height
    float *m_zbuffer; // width*height
    int m_width, m_height;
    void CreateBuffers(int width, int height);

public:
    // construction
    void Init();
    void DrawTriangles(vector<vec3>* vertices);
    void SetCameraTransform(mat4& cTransform);
    void SetProjection(mat4& projection);
    void SetObjectMatrix(mat4& oTransform, mat3& nTransform);
    void SwapBuffers();
}

class vec3 { 
public:
    GLfloat x;
    GLfloat y;
    GLfloat z;
    vec3( GLfloat s = GLfloat(0.0) ): 
        x(s), y(s), z(s) {}
    
    vec3( GLfloat x, GLfloat y, GLfloat z ): 
        x(x), y(y), z(z) {}
    ...
}
vec3 operator + ( const vec3& v ) const {
    return vec3( x + v.x, y + v.y, z + v.z );
}

vec3 operator * ( const GLfloat s ) const {
    return vec3( s*x, s*y, s*z );
}

vec3 operator * ( const vec3& v ) const {
    return vec3( x*v.x, y*v.y, z*v.z );
}
class mat2 {
    vec2 _m[2];
public:
    mat2( const GLfloat d = GLfloat(1.0) ){
        _m[0].x = d;  _m[1].y = d;
    }
    mat2( const vec2& a, const vec2& b ){
        _m[0] = a;  _m[1] = b;
    }
    mat2( GLfloat m00, GLfloat m10,
          GLfloat m01, GLfloat m11 ) {
        _m[0] = vec2( m00, m01 );
        _m[1] = vec2( m10, m11 );
    }
}
mat2 operator + ( const mat2& m ) const {
    return mat2( _m[0]+m[0], _m[1]+m[1] );
}
mat2 operator * ( const GLfloat s ) const {
    return mat2( s*_m[0], s*_m[1] );
}
mat2 operator * ( const mat2& m ) const {
    mat2  a( 0.0 );
    for ( int i = 0; i < 2; ++i )
        for ( int j = 0; j < 2; ++j )
            for ( int k = 0; k < 2; ++k )
                a[i][j] += _m[i][k] * m[k][j];
    return a;
}
Suggested Readings

• Interactive Computer Graphics, Chapter 3 and Appendices B and C.