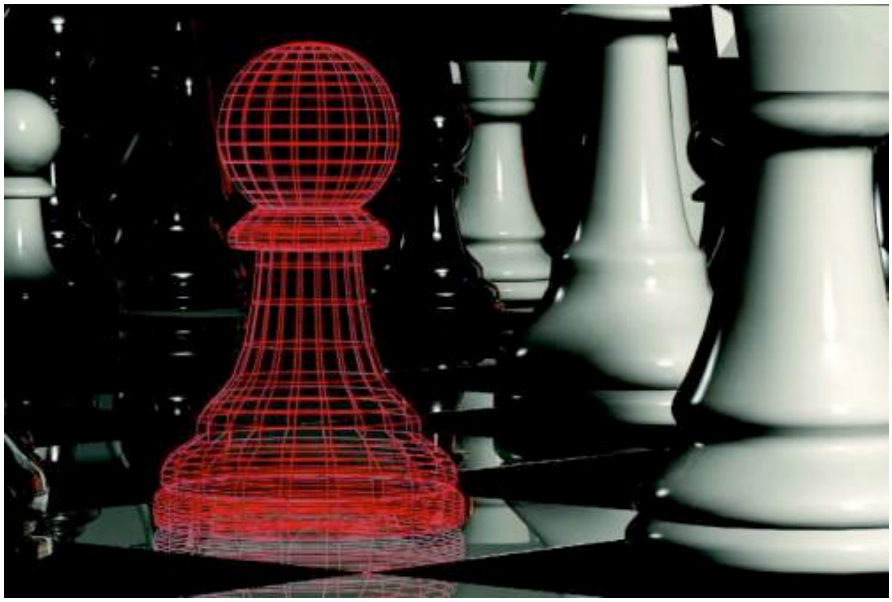
As to methods there may be a million and then some, but principles are few. The man who grasps principles can successfully select his own methods. The man who tries methods, ignoring principles, is sure to have trouble.
Emerson, Ralph Waldo

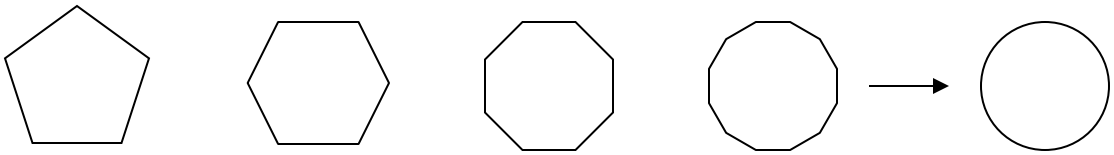
Tessellation



Tessellation: 2 Concepts Combined

Circle as polygon

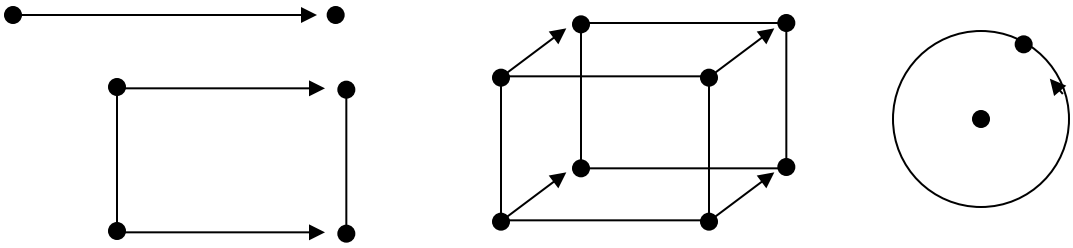
- A circle can be represented by a polygon with many sides (>15).



Vertices in motion

("Generative obj. description")

- Circle drawn by swinging (i.e., Rotate) a point at a fixed length around a center point



We'll Use Geometric Transformations

- For points written in homogeneous coordinates,

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

translation, scaling and rotation relative to the origin are expressed homogeneously as:

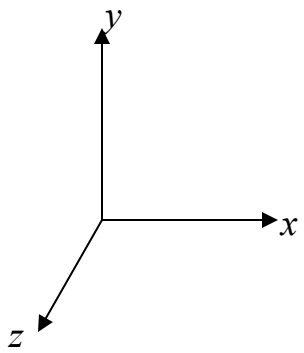
$$T(dx, dy) = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \quad v' = T(dx, dy)v$$

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad v' = S(s_x, s_y)v$$

$$R(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad v' = R(\phi)v$$

3D Basic Transformations (1/2)

(right-handed coordinate system)



- Translation $\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- Scaling $\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3D Basic Transformations (2/2)

(right-handed coordinate system)

- Rotation about X-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation about Y-axis

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation about Z-axis

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Important Digression: Row/Column Notation, Matrix Multiplication and Combining Transformations

Rule for Dot Product

- Also known as scalar product, or inner product. The result is a scalar (i.e., a number, not a vector)

- Defined as the sum of the pairwise multiplications

- Example:

$$\begin{bmatrix} a & b & c & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = ax + by + cz + dw$$

- Note, the result is not a vector of the component-wise multiplications, it is a scalar value

Matrix-Matrix Multiplication

- Generally, how do we compute the product MN from matrices M and N?

- One way to think of matrix multiplication is in terms of row and column vectors: MN_{ij} = the dot product of the i^{th} row of M and the j^{th} column of N

- **It is important to note that the rows of M and the columns of N must be the same size in order to compute their dot product**

- So for M, an *m x n* matrix, and N, an *n x k* matrix:

$$MN = \begin{bmatrix} row_{M,1} \bullet col_{N,1} & \cdots & \cdots & row_{M,m} \bullet col_{N,k} \\ \vdots & row_{M,i} \bullet col_{N,j} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ row_{M,m} \bullet col_{N,1} & \cdots & \cdots & row_{M,m} \bullet col_{N,k} \end{bmatrix}$$

- Where rowA,x means the xth row of A, and similarly, colA,x means the xth column of A

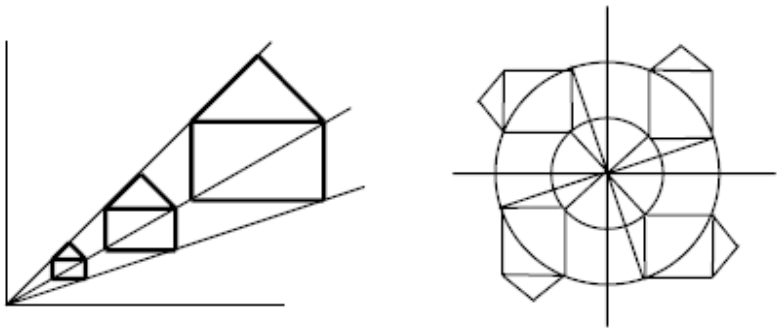
Correct or Incorrect? (aka Crimes Against Matrices)

Can we multiply (mind the order):

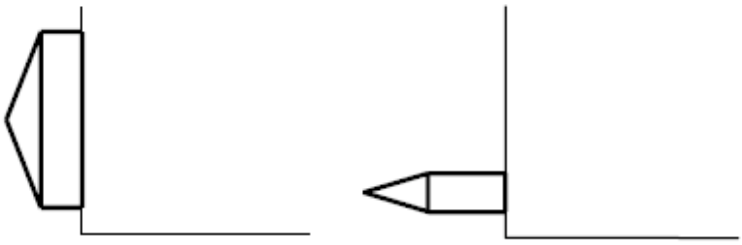
- A 4x4 matrix with a 4x1 matrix (i.e., column vector)?
- A 4x4 matrix with a 1x4 matrix (i.e., row vector)?
- A 1x4 matrix (i.e., row vector) with a 4x4 matrix?
- A 4x1 matrix (i.e., column vector) with a 4x4 matrix?

Combining Transformations

Matrices representing transformations performed in a sequence can be composed into a single matrix



Such combinations can be used to avoid unwanted Translation (unwittingly induced by scaling and rotation)



Left: R, followed by S
Right: S, followed by R

Order matters

Commutative and Non-Commutative Combinations of Transformations – Be Careful!

In 2D

- Commutative
  - Translate, translate
  - Scale, scale
  - Rotate, rotate
  - Scale uniformly, rotate
- Non-commutative
  - Non-uniform scale, rotate
  - Translate, scale
  - Rotate, translate

In 3D

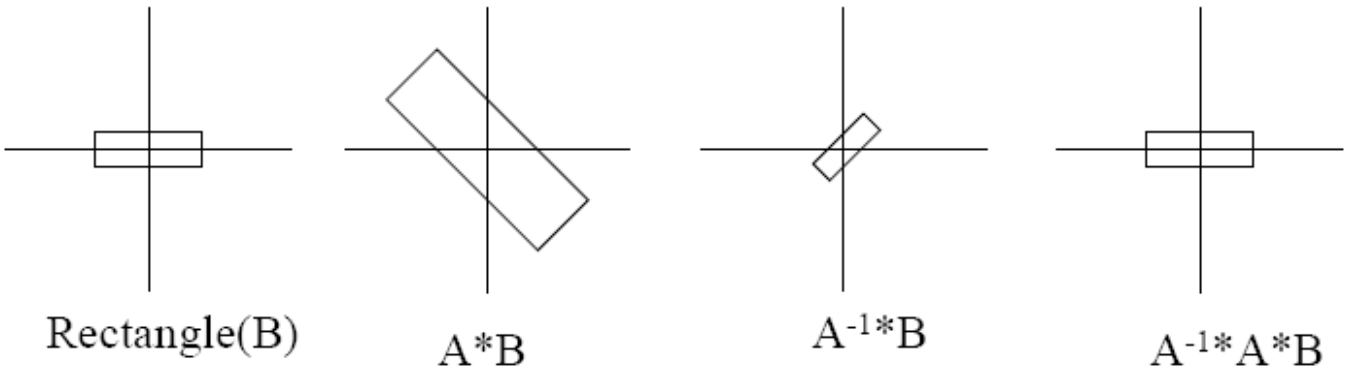
- Commutative
  - Translate, translate
  - Scale, scale
  - Scale uniformly, rotate
- Non-commutative
  - Non-uniform scale, rotate
  - Translate, scale
  - Rotate, translate
  - **Rotate, rotate**

What Does an Inverse Do?

- The inverse A^{-1} of transformation A will "undo" the result of transforming by A

- For example: if A scales by a factor of two and rotates 135°, then A^{-1} will rotate by –135° and then scale by one half

- Computing the inverse of a matrix: *Elementary Linear Algebra, by Howard Anton,* which can be found at the library



Correct or Incorrect?

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix}$$

Addendum – Matrix Notation

- The application of matrices in the row vector notation is executed in the reverse order of applications in the column vector notation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \leftrightarrow \begin{bmatrix} x & y & z \end{bmatrix}$$

- Column format: vector follows transformation matrix.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Row format: vector precedes matrix and is pos-multiplied by it.

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

But, There's a Problem…

- Notice that

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} ax+dy+gz & bx+ey+hz & cx+fy+iz \end{bmatrix}$$

- while

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax+by+cz \\ dx+ey+fz \\ gx+hy+iz \end{bmatrix}$$

Solution to Notational Problem

- In order for both types of notations to yield the same result, a matrix in the row system must be the transpose of the matrix in the column system
- Transpose is defined such that each entry at (i,j) in M, is mapped to (j,i) in its transpose (which is denoted M^T). You can visualize M^T as rotating M around its main diagonal

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, M^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

- Again, the two types of notation are equivalent:

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} = \begin{bmatrix} ax+by+cz & dx+ey+fz & gx+hy+iz \end{bmatrix} \leftrightarrow$$

$$\begin{bmatrix} ax+by+cz \\ dx+ey+fz \\ gx+hy+iz \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Different texts and graphics packages use different notations. Be careful!

Matrix Notation and Composition

- Application of matrices in row-major notation is reverse of application in column-major notation:

$$TRSv$$

$$\xleftarrow{\text{Matrices applied right to left}}$$

$$vS^TR^TT^T$$

$$\xrightarrow{\text{Matrices applied left to right}}$$

- In cs1566 we use the column-major notation

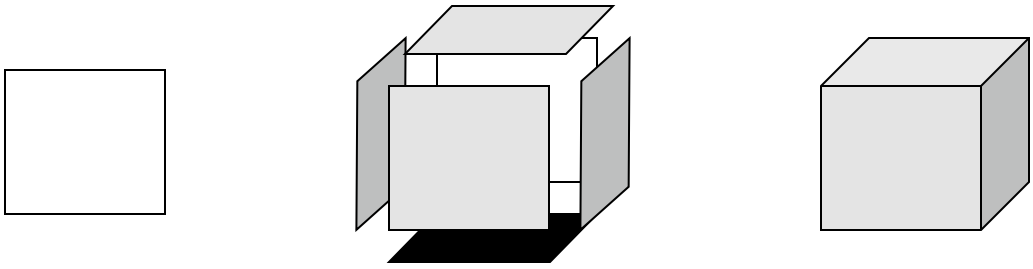
Are OpenGL Matrices Column-major or Row-major?

- For programming purposes, OpenGL matrices are 16-value arrays with base vectors laid out contiguously in memory. The translation components occupy the 13th, 14th, and 15th elements of the 16-element matrix.

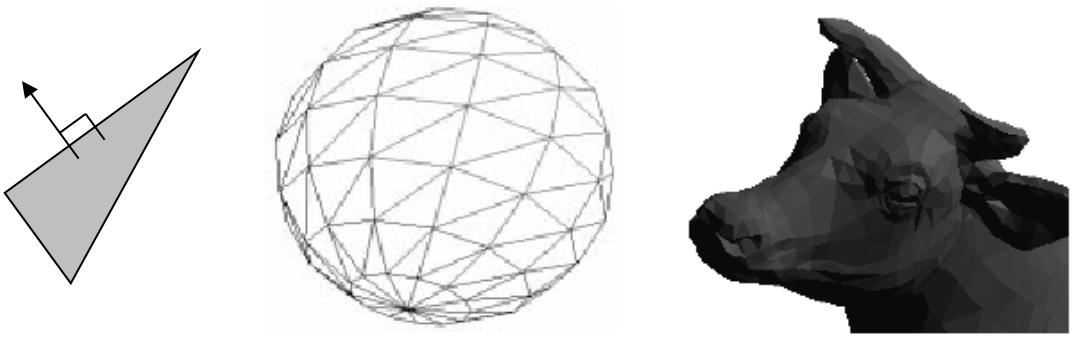
- Column-major versus row-major is purely a notational convention. Note that post-multiplying with column-major matrices produces the same result as pre-multiplying with row-major matrices.

- The OpenGL Specification and the OpenGL Reference Manual both use column-major notation. One can use any notation, as long as it's clearly stated.

Back to Tessellation



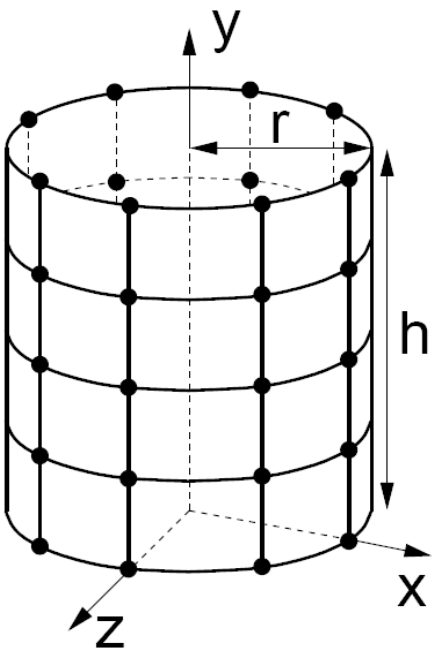
- Triangles and tri-meshes



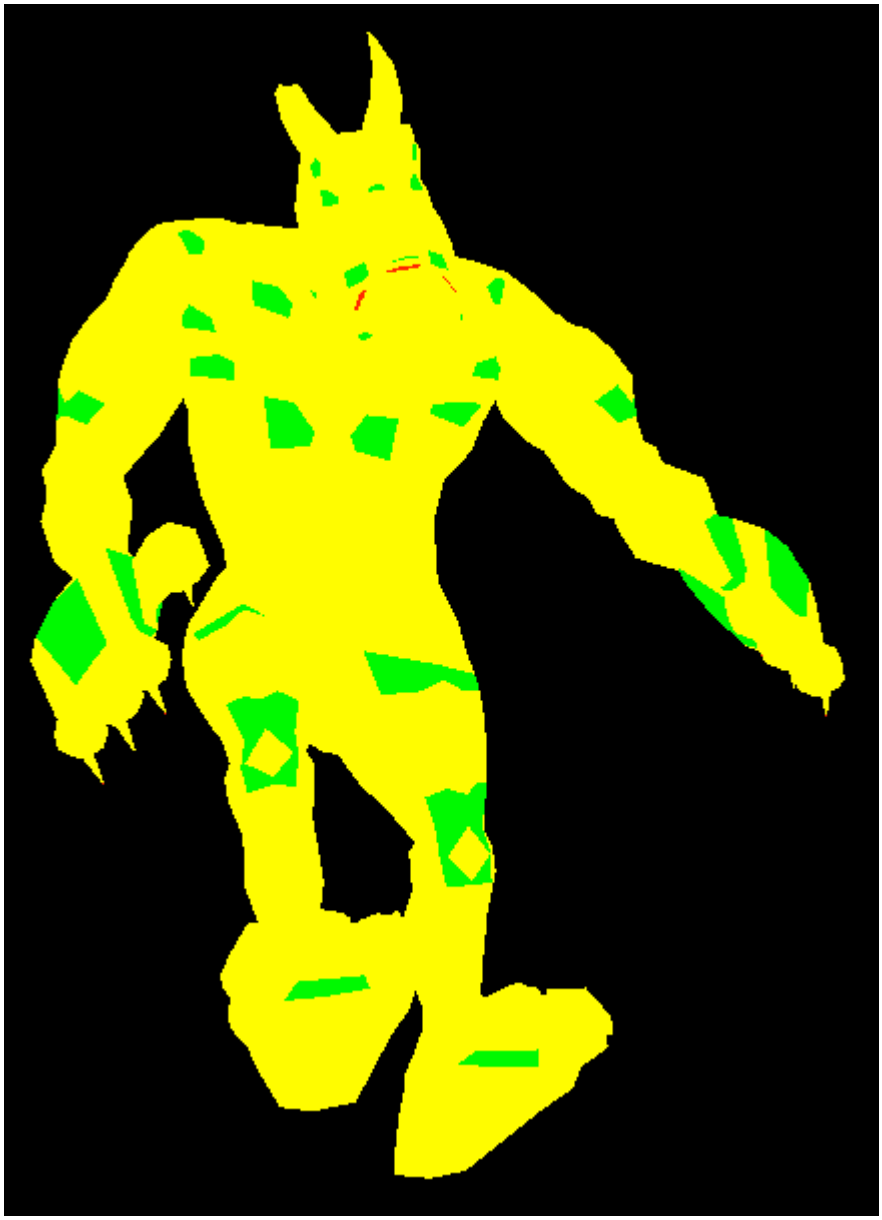
- Tessellation: collection of planar faces that form a 2D or 3D surface

- Challenges: defining the geometry (vertices), then stitching them into faces

Making a Cylinder



Meshes



The Wavefront and Java3D *.obj* Format

- # some text
  - Line is a comment until the end of the line
- v *float float float*
  - A single vertex's geometric position in space. The first vertex listed in the file has index 1, and subsequent vertices are numbered sequentially.
- vt *float float*
  - A texture coordinate. The first texture coordinate in the file is index 1, and subsequent textures are numbered sequentially. The number of texture coordinate does not need to match the number of vertices.
- vn *float float float*
  - A vertex normal. The first normal in the file is index 1, and subsequent normals are numbered sequentially.
- f *int int int* ...
  - **or**
- f *int/int int/int int/int* . . .
  - **or**
- f *int/int/int int/int/int int/int/int* ...
  - A polygonal face. The numbers are indexes into the arrays of vertex positions, texture coordinates, and normals respectively. A number may be omitted if, for example, texture coordinates are not being defined in the model. There is no maximum number of vertices that a single polygon may contain. The .obj file specification says that each face must be flat and convex.

Example

- The example file sample.obj is given below (it's a cube):

```
v 1 1 1
v 1 1 -1
v 1 -1 1
v 1 -1 -1
v -1 1 1
v -1 1 -1
v -1 -1 1
v -1 -1 -1
f 1 3 4 2
f 5 7 8 6
f 1 5 6 2
f 3 7 8 4
f 1 5 7 3
f 2 6 8 4
```