Introduction to Computer Graphics with OpenGL

Ioannis Tsiombikas

nuclear@siggraph.org

Computer graphics

Algorithms to transform mathematical representations of 3D environments to images.

Possible representations:

Computer graphics

Algorithms to transform mathematical representations of 3D environments to images.

Possible representations:

- Polyhedral approximation of surfaces.

Computer graphics

Algorithms to transform mathematical representations of 3D environments to images.

Possible representations:

- Polyhedral approximation of surfaces.
- Mathematical equations describing surfaces (i.e. $x^2 + y^2 + z^2 = r^2$).

Computer graphics

Algorithms to transform mathematical representations of 3D environments to images.

Possible representations:

- Polyhedral approximation of surfaces.
- Mathematical equations describing surfaces (i.e. $x^2 + y^2 + z^2 = r^2$).
- Volume defined by density values (binary or not) at discrete points in a 3D scalar field (voxels).

Real–time graphics

The major distinction in graphics: real–time vs off–line rendering.

Real–time graphics algorithms sacrifice image quality to achieve rapid, sub–second, drawing rates. This enables us to interactively rearrange objects or the view–point thus allowing us to “navigate” in a 3D environment or manipulate it.

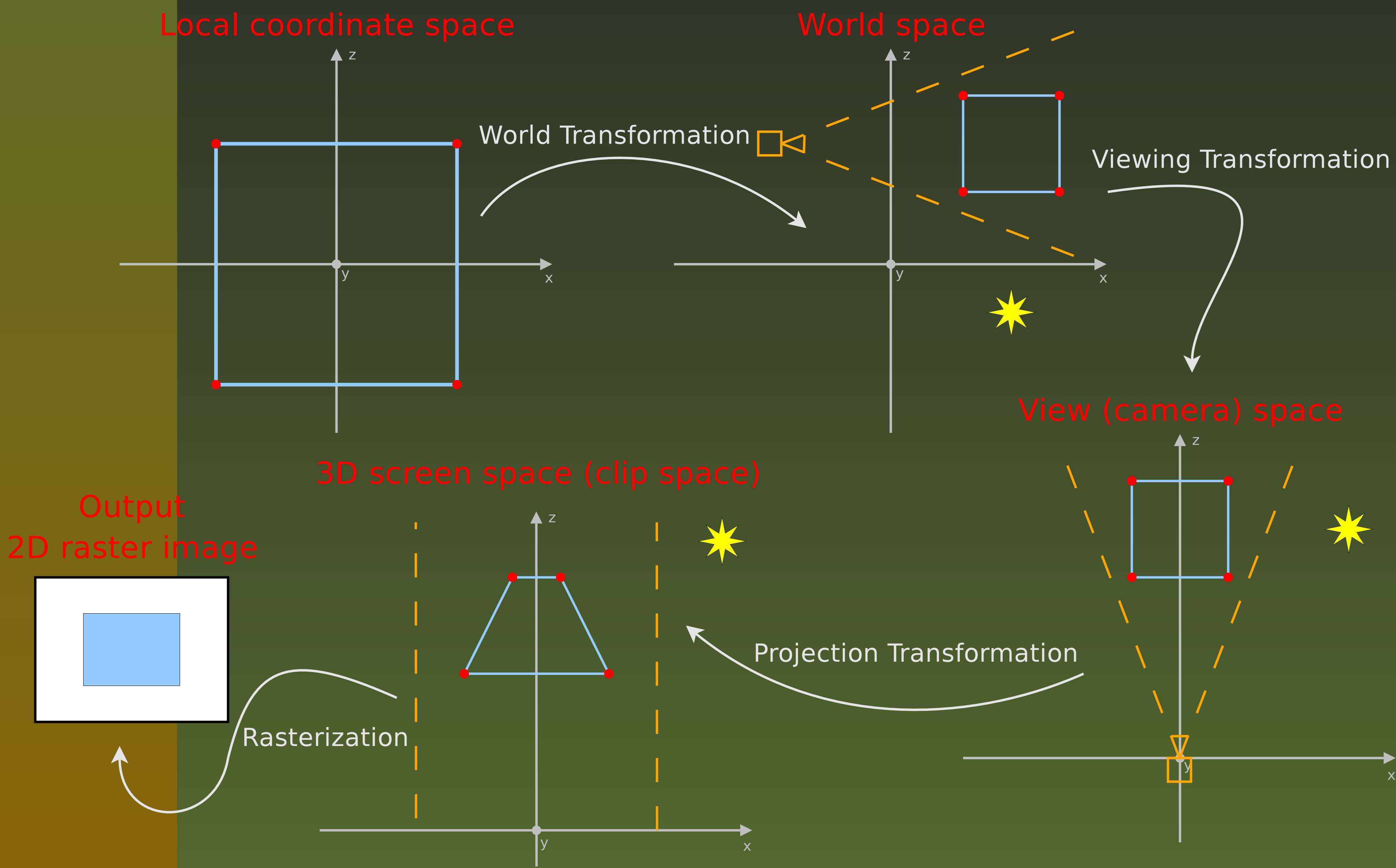
Used in games, interactive visualizations, 3D modelling/animation tools, etc.

OpenGL

OpenGL is an open standard for dealing with 3D graphics, with source–compatible implementations on every major platform capable of graphical output.

- Controlled by a special committee, the Architecture Review Board (ARB).
- Targeted towards interactive programs and real–time graphics.
- Stable programming interface.
- Flexible due to an extension mechanism for additional "cutting–edge" functionality.
- Simple state–machine design.

The rendering pipeline



Transformations

A 3x3 matrix defines a linear transformation in 3D space (rotation, scaling, etc.). However it is more convinient to work on 4D space and at the end keep a 3D projection of that.

Our vectors become (x, y, z, w) with $w = 1$ for the equivalent of 3D vectors, and we use 4x4 matrices to transform them.

By using this technique (called homogeneous coordinates) we can place points at infinity ($w = 0$), but most importantly, include 3D translation in our transformation matrices.

Transformations

Transform vectors by multiplying them with the appropriate matrix.

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

To concatenate a series of transformations in one matrix, multiply all the matrices together.
Note: *order matters*! Matrix multiplication is not commutative.

Transformations: rotation

$$Rot_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Rot_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Rot_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformations: translation/scaling

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OpenGL transformations

OpenGL maintains matrix stacks for all stages of the pipeline. However world and view transformations are combined in one (modelview).
To manipulate the matrix state, first specify which stack we wish to affect with *glMatrixMode()*, and call:

OpenGL transformations

OpenGL maintains matrix stacks for all stages of the pipeline. However world and view transformations are combined in one (modelview).
To manipulate the matrix state, first specify which stack we wish to affect with *glMatrixMode()*, and call:

- *glLoadMatrixf()* / *glMultMatrixf()* to load or concatenate an arbitrary matrix to the top matrix.

OpenGL transformations

OpenGL maintains matrix stacks for all stages of the pipeline. However world and view transformations are combined in one (modelview).
To manipulate the matrix state, first specify which stack we wish to affect with *glMatrixMode()*, and call:

- *glLoadMatrixf()* / *glMultMatrixf()* to load or concatenate an arbitrary matrix to the top matrix.
- *glLoadIdentity()* to load the identity matrix.

OpenGL transformations

OpenGL maintains matrix stacks for all stages of the pipeline. However world and view transformations are combined in one (modelview).
To manipulate the matrix state, first specify which stack we wish to affect with *glMatrixMode()*, and call:

- *glLoadMatrixf()* / *glMultMatrixf()* to load or concatenate an arbitrary matrix to the top matrix.
- *glLoadIdentity()* to load the identity matrix.
- *glTranslatef()* / *glRotatef()* / *glScalef()* to concatenate the desired transformation matrix to the top matrix.

OpenGL transformations

OpenGL maintains matrix stacks for all stages of the pipeline. However world and view transformations are combined in one (modelview).
To manipulate the matrix state, first specify which stack we wish to affect with *glMatrixMode()*, and call:

- *glLoadMatrixf()* / *glMultMatrixf()* to load or concatenate an arbitrary matrix to the top matrix.
- *glLoadIdentity()* to load the identity matrix.
- *glTranslatef()* / *glRotatef()* / *glScalef()* to concatenate the desired transformation matrix to the top matrix.
- *glPushMatrix()* / *glPopMatrix()* for the usual stack operations.

OpenGL 3D object data

Vertices, grouped in triangles, quadrilaterals, or polygons define the surfaces of objects in 3D space. Apart from their positions that define the surface, there is a number of additional per–vertex data commonly given to OpenGL:

- Vertex colors (if lighting is disabled, useful for precalculated lighting).
- Normal vectors (used for lighting calculations).
- Texture mapping coordinates.

OpenGL 3D object data

Vertex data can be given to OpenGL in many ways.

OpenGL 3D object data

Vertex data can be given to OpenGL in many ways.

- Immediate mode, *glBegin()* / *glEnd()*.

OpenGL 3D object data

Vertex data can be given to OpenGL in many ways.

- Immediate mode, *glBegin()* / *glEnd()*.
- Vertex arrays (in GL client memory).

OpenGL 3D object data

Vertex data can be given to OpenGL in many ways.

- Immediate mode, *glBegin()* / *glEnd()*.
- Vertex arrays (in GL client memory).
- Vertex buffer objects (vertex arrays in GL server memory).

OpenGL 3D object data

Vertex data can be given to OpenGL in many ways.

- Immediate mode, *glBegin() / glEnd()*.
- Vertex arrays (in GL client memory).
- Vertex buffer objects (vertex arrays in GL server memory).
- Display lists.

Lighting

For each vertex, a color is calculated as a function of the intensity of the illumination reflected off the surface towards the viewpoint, and the *material* of the surface. Then color from each polygon's vertices are linearly interpolated across its surface to calculate the color of each pixel. Lights are represented as points, typically in world coordinates.

Lighting

For each vertex, a color is calculated as a function of the intensity of the illumination reflected off the surface towards the viewpoint, and the *material* of the surface. Then color from each polygon's vertices are linearly interpolated across its surface to calculate the color of each pixel. Lights are represented as points, typically in world coordinates.

$$A\ M_a + \sum_{i=1}^{lnum} D(l_i, n)\ M_d + S(l_i, v)\ M_s$$

Lighting

For each vertex, a color is calculated as a function of the intensity of the illumination reflected off the surface towards the viewpoint, and the *material* of the surface. Then color from each polygon's vertices are linearly interpolated across its surface to calculate the color of each pixel. Lights are represented as points, typically in world coordinates.

$$A\, M_a + \sum_{i=1}^{lnum} D(l_i, n)\, M_d + S(l_i, v)\, M_s$$

$$D(l, n) = l \cdot n$$

Lighting

For each vertex, a color is calculated as a function of the intensity of the illumination reflected off the surface towards the viewpoint, and the *material* of the surface. Then color from each polygon's vertices are linearly interpolated across its surface to calculate the color of each pixel. Lights are represented as points, typically in world coordinates.

$$A\,M_a + \sum_{i=1}^{lnum} D(l_i, n)\,M_d + S(l_i, v)\,M_s$$

$$D(l, n) = l \cdot n$$

$$S(l, n, v, p) = (l \cdot reflect(v, n))^p$$

OpenGL Lighting: Materials

OpenGL handles light calculations if we provide vertex normals, light and material parameters, and enable set the appropriate state.

The following material parameters can be set using the *glMaterialf()* and *glMaterialfv()* functions:

OpenGL Lighting: Materials

OpenGL handles light calculations if we provide vertex normals, light and material parameters, and enable set the appropriate state.

The following material parameters can be set using the *glMaterialf()* and *glMaterialfv()* functions:

- Color (seperate for ambient, specular and diffuse).

OpenGL Lighting: Materials

OpenGL handles light calculations if we provide vertex normals, light and material parameters, and enable set the appropriate state.

The following material parameters can be set using the *glMaterialf()* and *glMaterialfv()* functions:

- Color (seperate for ambient, specular and diffuse).
- Shininess (the specular power).

OpenGL Lighting: Materials

OpenGL handles light calculations if we provide vertex normals, light and material parameters, and enable set the appropriate state.

The following material parameters can be set using the *glMaterialf()* and *glMaterialfv()* functions:

- Color (seperate for ambient, specular and diffuse).
- Shininess (the specular power).
- Self–illumination.

OpenGL Lighting: Lights

Light parameters are set using the *glLightf()* and *glLightfv()*.

Light parameters:

OpenGL Lighting: Lights

Light parameters are set using the *glLightf()* and *glLightfv()*.

Light parameters:

- Position or direction.

OpenGL Lighting: Lights

Light parameters are set using the *glLightf()* and *glLightfv()*.

Light parameters:

- Position or direction.
- Color (seperated into ambient, diffuse and specular components).

OpenGL Lighting: Lights

Light parameters are set using the *glLightf()* and *glLightfv()*.

Light parameters:

- Position or direction.
- Color (seperated into ambient, diffuse and specular components).
- Optional spotlight illumination cone.

OpenGL Lighting: Lights

Light parameters are set using the *glLightf()* and *glLightfv()*.

Light parameters:

- Position or direction.
- Color (seperated into ambient, diffuse and specular components).
- Optional spotlight illumination cone.
- Optional distance attenuation coefficients.