What is Interactive Computer Graphics?

Example: Computer Aided Design (CAD)

- interactive modeling (simple visual representations)
- metrically correct constructions as a basis for automated manufacturing
- results: Technical sketches, simple rendered still images

Example: Computer Graphics Animation

- modeling & design of objects
- modeling of dynamic behavior
- results: HQ static pictures and static movies

Example: Interactive Computer Graphics

- on-line rendering (real-time!)
- rendering as realistic as frame-rate allows
- results: user driven non static 3D scenario
What is Interactive Computer Graphics?

Example: Interactive Computer Graphics

Interactive Graphical Systems: Structure

- Multimedia Feedback
- Multimodal Input
- Real-time

Computer-simulation

Replication

Real world

How are graphical objects defined?

Surface representations

=> infinite number of points to form object surface (mathematical functions)

=> finite set of surface points
   - rendered as point clouds
   - rendered as polygons given a certain connectivity

=> finite set of volume points (voxels)
   - rendered as point clouds
   - rendered as solid volume objects

Definition of “points”

Based on the definition of a reference coordinate system by their components

1) Cartesian coordinate system \{ O; i, j, k \}
   - orthogonal
   - orthonormal
2) polar co-ordinate system \{ O; r, \lambda, \phi \}

definition of O, y, and z

![Diagram of polar coordinate system]

\[ r = \sqrt{x^2 + y^2 + z^2} \]

\[ \tan \lambda = \frac{x}{z} \quad -\pi < \lambda \leq \pi \]

\[ -\pi/2 < \phi \leq \pi/2 \]

Relations between polar and cartesian co-ordinates

x = r \cdot \cos \phi \cdot \cos \lambda

y = r \cdot \sin \phi

z = r \cdot \cos \phi \cdot \sin \lambda

Vector algebra (quick rehearsal of useful formulas)

1. Inner product / dot product of two vectors

\[ a \cdot b = a_x b_x + a_y b_y + a_z b_z \]

\[ |a| = \sqrt{a \cdot a} = \sqrt{a_x^2 + a_y^2 + a_z^2} \]

\[ \alpha = \cos^{-1} \frac{a \cdot b}{|a| \cdot |b|} \]

\[ |b_a| = a \cdot b \wedge |a| = 1 \]

Used for

a) calculation of length and distances
b) angle measurements

Application of the dot product - Backface Culling

Given: A polygon vertex P, and polygon normal vector N

Users viewing position V

Question: Is the polygon facing towards the observer?

Solution: The polygon is facing towards the observer if the normal vector of that face is facing towards the observer. This is the case, if the length of the projection of the vector V' onto N is greater than zero. See next slide for two examples.
Application of the dot product - Backface Culling

Let: \[ P = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad N = \begin{pmatrix} 2/3 \\ 1/3 \\ 2/3 \end{pmatrix}, \quad |N| = \sqrt{4/9 + 1/9 + 4/9} = \sqrt{9/9} = 1 \]

Case 1: Let \[ V = \begin{pmatrix} 10 \\ 7 \\ 15 \end{pmatrix} \]
\[ V' = V - P = \begin{pmatrix} 10 - 1 \\ 7 - 1 \\ 15 - 0 \end{pmatrix} = \begin{pmatrix} 9 \\ 6 \\ 15 \end{pmatrix} \]
\[ |v| = N \cdot V' = \frac{2}{3} \cdot 9 + \frac{1}{3} \cdot 6 + \frac{2}{3} \cdot 15 = 6 + 2 + 10 = 18 \]
18 > 0 -> The polygon is facing towards \( V \).

Case 2: Let \[ V = \begin{pmatrix} -5 \\ 7 \\ -9 \end{pmatrix} \]
\[ V' = V - P = \begin{pmatrix} -5 - 1 \\ 7 - 1 \\ -9 - 0 \end{pmatrix} = \begin{pmatrix} -6 \\ 6 \\ -9 \end{pmatrix} \]
\[ |v| = N \cdot V' = \frac{2}{3} \cdot -6 + \frac{1}{3} \cdot 6 + \frac{2}{3} \cdot -9 = -4 + 2 - 6 = -8 \]
-8 < 0 -> The polygon is **not** facing towards \( V \).

Application of the cross product - Polygon Normal Setup

Given: A triangle given by vertices A, B, and C

Question: How does the polygon's normal vector look like?

\[ A = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \quad B = \begin{pmatrix} 2 \\ 15 \\ -1 \end{pmatrix}, \quad C = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \]

Span vectors:

\[ AB = B - A = \begin{pmatrix} 1 - 2 \\ 15 - 1 \\ -1 - 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.5 \\ -1 \end{pmatrix}, \quad AC = C - A = \begin{pmatrix} 0 \\ 2 - 1 \\ -1 - 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}, \quad a \times c = \begin{pmatrix} 0.5 \\ 1 \\ -1 \end{pmatrix} \]

\[ N = a \times c = \begin{pmatrix} 0.5 \\ 1 \\ -1 \end{pmatrix} \]

Vector algebra (quick rehearsal of useful formulas)

2. Vector product or cross product of two vectors

\[ a \times b = c \]
\[ = (a_x \cdot b_z - b_y \cdot a_z) \hat{i} + (a_y \cdot b_x - b_z \cdot a_x) \hat{j} + (a_z \cdot b_y - b_x \cdot a_y) \hat{k} \]
\[ = \begin{pmatrix} a_x \cdot b_z - b_y \cdot a_z \\ a_y \cdot b_x - b_z \cdot a_x \\ a_z \cdot b_y - b_x \cdot a_y \end{pmatrix} \]
\( c \) is a vector perpendicular to \( a \) and \( b \).

Used for:
- a) calculation of surface normal vectors
- b) test for collinearity of two vectors (\( c = 0 \))
- c) calculation of reference frames

Concatenation of Transforms: Matrix Multiplication

Matrix \( A \) ; \( n \times k \) elements

Matrix \( B \) ; \( k \times l \) elements

=> \( C = AB \) defined; and the resulting matrix \( C \) has \( n \times l \) elements

=> generally: \( AB \not= BA \)

\[ c_{ij} = \sum_{s=1}^{k} a_{is} b_{sj}; \quad 1 < i < n, 1 < j < l \]
Transformation of Points: Matrix-Vector Multiplication

Transformation represented as a $M_{n,k}$ where $n=4$, $k=4$.

Point represented by homogeneous coordinates $\{x, y, z, 1\}$ as column vector $P$ where $k = 4$, $l = 1$.

The resulting matrix has $n=4$ rows and $l=1$ columns

$$ P' = M \cdot P = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + m_{13}z + m_{14} \\ m_{21}x + m_{22}y + m_{23}z + m_{24} \\ m_{31}x + m_{32}y + m_{33}z + m_{34} \\ m_{41}x + m_{42}y + m_{43}z + m_{44} \end{bmatrix} $$

Matrix representation of basic geometric transforms

<table>
<thead>
<tr>
<th>Scale</th>
<th>Translation</th>
<th>Rotations</th>
</tr>
</thead>
</table>
| \[
\begin{bmatrix}
 s_x & 0 & 0 & 0 \\
 0 & s_y & 0 & 0 \\
 0 & 0 & s_z & 0 \\
 0 & 0 & 0 & 1
\end{bmatrix}
| \[
\begin{bmatrix}
 1 & 0 & 0 & t_x \\
 0 & 1 & 0 & t_y \\
 0 & 0 & 1 & t_z \\
 0 & 0 & 0 & 1
\end{bmatrix}
| \[
\begin{bmatrix}
 \cos \theta & -\sin \theta & 0 & 0 \\
 \sin \theta & \cos \theta & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 1
\end{bmatrix}
| \[
\begin{bmatrix}
 \cos \theta & 0 & \sin \theta & 0 \\
 0 & 1 & 0 & 0 \\
 -\sin \theta & 0 & \cos \theta & 0 \\
 0 & 0 & 0 & 1
\end{bmatrix}

Accumulated/composite transformations

Any attitude of an object can be expressed/decomposed as an accumulation of elementary transforms.

Example: $A = R_zTS$

Observe: Matrix concatenation order (transformation sequence) is important.

Example: $A = R_zT$

Example: $A = TR_z$

Accumulated/composite transformations

A) 1. Rotate about $Z$, 50 deg.  
2. Translate along $X$, 0.5  

B) 1. Rotate about $X$, 25 deg.  
2. Translate along $X$, 0.5  
3. Rotate about $Z$, 50 deg.
Calculation of Reference Frames (Attitude Matrix)

Given: Three points in space identify the spatial attitude of an object’s reference frame

A, B, and C can be used to calculate a new orthonormal basis {a, b, c}

\[ a = C - A \]
\[ b = (B - A) \times a \]
\[ c = a \times b \]

Note: a, b, and c must be normalized!

A transformation matrix \( T_{ab} \) rotates the unit vector (1,0,0,1) into the new base vector \( a=(ax, ay, az, 1) \)

\[
T_{ab} = \begin{bmatrix}
1 & t_{12} & t_{13} & t_{14} \\
t_{21} & 1 & t_{23} & t_{24} \\
t_{31} & t_{32} & 1 & t_{34} \\
t_{41} & t_{42} & t_{43} & 1
\end{bmatrix}
\]

\[ T_{ab} \cdot x = \begin{bmatrix}
1 & t_{12} & t_{13} & t_{14} \\
t_{21} & 1 & t_{23} & t_{24} \\
t_{31} & t_{32} & 1 & t_{34} \\
t_{41} & t_{42} & t_{43} & 1
\end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}
\]

Hence: \( T_{ab} = \begin{bmatrix} a_x & ? & ? & 0 \\ a_y & ? & ? & 0 \\ a_z & ? & ? & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \)

Calculation of Reference Frames (Attitude Matrix)

\[
T_{oa} = \begin{bmatrix}
t_{11} & t_{12} & t_{13} & t_{14} \\
t_{21} & t_{22} & t_{23} & t_{24} \\
t_{31} & t_{32} & t_{33} & t_{34} \\
t_{41} & t_{42} & t_{43} & t_{44}
\end{bmatrix}
\]

\[ T_{oa} \cdot O = \begin{bmatrix}
t_{11} & t_{12} & t_{13} & t_{14} \\
t_{21} & t_{22} & t_{23} & t_{24} \\
t_{31} & t_{32} & t_{33} & t_{34} \\
t_{41} & t_{42} & t_{43} & t_{44}
\end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}
\]

Hence: \( T_{oa} = \begin{bmatrix} a_x & ? & ? & A_x \\ a_y & ? & ? & A_y \\ a_z & ? & ? & A_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \)

Since \( T_{oa} \), \( T_{by} \), and \( T_{cz} \) operate independently on the respective base vectors, they can be combined into a single transformation matrix, that rotates all base vectors simultaneously. The translation matrix operates independently on to translate the origin, regardless of the rotational components. Hence, the combined transformation matrix can be denoted as:

\[
T = \begin{bmatrix}
a_x & b_x & c_x & A_x \\
a_y & b_y & c_y & A_y \\
a_z & b_z & c_z & A_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Calculation of Reference Frames (Attitude Matrix)

Representations of 3D Models

Quantitative Models

Mathematical, physical, chemical... function define vertex data

Procedural Models

Declarative or procedural system describes vertex data and connectivity
(rule based, language based, recursive and fractal functions)

Hierarchical Geometric Models

Explicit definition of objects by vertex data and connectivity
Re-Usage of geometric prototypes
Defining hierarchies and geometric relations between instances
How are graphical objects defined?

Mathematical functions: Example MATLAB

Finite set of volume points: Voxelviewer

Finite set of surface points: Submarine

Object representations in interactive CG

Graphic rendering engines are traditionally optimized for:
- drawing of 3D points
- drawing of 3D vectors
- drawing of 3D polygons (mostly triangles)
- (some can draw spheres e.g. SGI Extreme)

=> objects are represented as composite clumps of vertices and polygons.

=> more advanced, analytical descriptions of objects are usually not found in interactive computer graphics (see 3D animation programs and raytracing software)
Object representations in *interactive* CG

Example: Scene with a teapot

1. explicit definition of vertices & polygons
   rendered using flat shading

2. combined approach: explicit vertex data and
   mathematical polynomial functions define
   curved surfaces.

---

Explicit geometry representations in *interactive* CG

**Organization of geometry data:**

<table>
<thead>
<tr>
<th>Vertex List</th>
<th>Polygon List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Data</td>
<td>Normal Vectors</td>
</tr>
<tr>
<td>Cartesian Coordinates</td>
<td>Texture Coordinates</td>
</tr>
<tr>
<td>Color Values</td>
<td>Other data e.g. Texture Reference</td>
</tr>
</tbody>
</table>

**Objects reference frame**

---

**Rendering Methods**

- **Point Clouds**
  - Object vertices are drawn as singular pixels
  - Color of the pixels can be constant (object color)
    or be shaded (depending on lights and object colors)
  - Determination of discrete screen-coordinates from
    model vertex coordinates
  - Very fast rendering method

Examples:

- ![Point Cloud Example 1](image1)
- ![Point Cloud Example 2](image2)
- ![Point Cloud Example 3](image3)

- **Wire Frames**
  - Edges of object polygons are drawn as lines
  - Color of the lines can be constant (object color)
    or be shaded (depending on lights and object colors)
  - Determination of discrete screen-coordinates for
    model vertices
  - 2D line drawing (scan conversion) between the vertices
  - Very fast rendering method if used without hidden
    surface removal

Examples:

- ![Wire Frame Example 1](image1)
- ![Wire Frame Example 2](image2)
- ![Wire Frame Example 3](image3)
Rendering Methods

- **Surface Shading**
  - Object polygons are drawn as filled areas
  - Color of the polygons can be constant (object color) or be shaded (depending on lights and object colors) (constant or ambient shading, diffuse shading)
  - Determination of discrete screen-coordinates for model vertices
  - 2D line drawing (scan conversion) of the edges
  - 2D area fill inside the polygon
  - Can be very slow depending on shading method

Examples:

![Example images](image1.png) ![Example images](image2.png)

Surface Shading Models

- **Constant / Flat Shading**
  - One shading value is calculated per polygon
  - Entire polygon is filled only with that color value
  - Fast but not nice looking

- **Gouraud Shading (Color Interpolation)**
  - Shading values are calculated for each vertex
  - Color values for each pixel inside polygon are linearly interpolated
  - Good visual appearance at cost of reduced rendering speed

- **Phong Shading (Normal Vector Interpolation)**
  - For each pixel the surface normal vector is interpolated from the given vertex normals
  - Shading calculation is performed for each pixel
  - Extremely slow rendering, very good visual results (specular properties)

Example: Flat Shading vs. Gouraud Shading

![Example images](image3.png) ![Example images](image4.png)

Requirements: Flat Shading vs. Gouraud Shading

**Flat-Shading:**

One surface normal vector per face/polygon.

**Phong-Shading:**

Surface normal vectors per vertex.
Usually calculated from surrounding polygon normal vectors.

Example: \( n_1 = (f_{n_1} + f_{n_2} + f_{n_3} + f_{n_4})/4 \)
Gouraud Shading and Phong Shading

**Gouraud**

Illumination and color calculation at vertices:

\[ C_i = (\mathbf{n} \cdot \mathbf{l}) \hat{\mathbf{n}} \]

Color calculation between vertices by (bi)linear interpolation:

\[ C_i = (1 - \alpha) \cdot C_i + \alpha \cdot C_j \land [0 \leq \alpha \leq 1] \]

More on illumination: see Foley, chapter 16.1

**Phong**

Illumination and color calculation for each pixel analytical:

\[ C_i = (\mathbf{n} \cdot \mathbf{l}) \hat{\mathbf{n}} \]

For pixels between vertices, interpolation of normal vector and calculation of illumination and color:

\[ n_i = (1 - \alpha) \cdot n_i + \alpha \cdot n_j \land [0 \leq \alpha \leq 1] \]

\[ C_i = (\mathbf{n} \cdot \mathbf{l}) \hat{\mathbf{n}} \]

Example:

\[ I_i = (n_i \cdot l) \hat{\mathbf{n}} \]

\[ l = 10, \mathbf{\xi} = -10 \]

\[ \lambda = 4, \mathbf{\gamma} = -4 \]

Differences: Gouraud Shading vs. Phong Shading

Explicit geometry representations in *interactive* CG

Organization of geometry data:

- **Vertex List**
  - Vertex Index
  - Cartesian Coordinates
  - Normal Vectors
  - Texture Coordinates
  - Color Values

- **Polygon List**
  - Polygon Index
  - Number of Vertices
  - List of Vertex Indices
  - Other data e.g. Texture Reference

Hierarchical modeling

**Purposes:**

- Construct complex objects in modular fashion
- Reusability of building blocks
- Ease of modeling
- Increase storage efficiency
- Build once
- Execute/re-use many times (display lists)
- Allow easy update of components
- Allow for easier animation of behavior
Hierarchical modeling

Example: simple robot

Three different generic objects

Several instances of objects arranged in hierarchic order

Creating the Model Hierarchy

```
base = VRT_NodeNew(root,"Robot Base");
trunk = VRT_NodeNew(base,"Robot Trunk");
head = VRT_NodeNew(trunk,"Robot Head");
left_arm = VRT_NodeNew(trunk,"Left Arm");
right_arm = VRT_NodeNew(trunk,"Right Arm");
cylinder = VRT_Cone(1.0,1.0,1.0,20);
cube = VRT_Cube(1);
box = VRT_Box(0.2,1.0,0.2);
VRT_NodeSetGeometry(base,cylinder);
VRT_NodeSetGeometry(trunk,cube);
VRT_NodeSetGeometry(head,cube);
VRT_NodeSetGeometry(left_arm,box);
VRT_NodeSetGeometry(right_arm,box);
```

Creating the Geometric Context

```
VRT_NodeScale(base,0.1,0.2,0.1);
VRT_NodeTranslate(trunk,0.0,2.0,0.0);
VRT_NodeScale(trunk,1.5,1.0,1.5);
VRT_NodeTranslate(head,0.0,1.2,0.0);
VRT_NodeScale(head,0.5,0.5,0.5);
VRT_NodeTranslate(right_arm,-1.4,0.8,-0.2)
VRT_NodeScale(right_arm,2.0,2.0,2.0);
VRT_NodeRotate(right_arm,180,0,0);
VRT_NodeTranslate(left_arm,1.0,0.8,-0.2);
VRT_NodeScale(left_arm,2.0,2.0,2.0);
VRT_NodeRotate(left_arm,180,0,0);
```