Interactive Graphical Systems
HT2005

Lesson 2 : Graphics Primer

Stefan Seipel
Key issues of this lecture

Representations of 3D models

Repetition of basic and useful math concepts for use in ICG

Establishing reference frames and transformation between reference frames

Handling of 3D models by the rendering hardware (fixed function pipeline, programmable architectures)

Handling of 3D models by the application developer
Interactive Graphical Systems: Structure

- Multimedia Feedback
- Multimodal Input
- Computer-simulation
- Real-time
- Replication
- Real world

Stefan Seipel, Department of Information Technology, Uppsala University
What is Interactive Computer Graphics?

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

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

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: Computer Aided Design (CAD)

Richard Nordhaus
School of Architecture and Planning
University of New Mexico

Stefan Seipel, Department of Information Technology, Uppsala University
What is Interactive Computer Graphics?

Example: Computer Graphics Animation

*Tin Toy*
Copyright Pixar Animation Studios
What is Interactive Computer Graphics?

Example: Interactive Computer Graphics

Design and Simulation of a Robot Cell
Copyright Sandia National Libraries
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 co-ordinate system by their components

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

definition of \( O, y, \) and \( z \)

\[-\pi < \lambda <= \pi\]

\[-\pi/2 < \phi <= \pi/2\]
Relations between polar and cartesian co-ordinates

\[ r = \sqrt{x^2 + y^2 + z^2} \]
\[ \tan \lambda = \frac{x}{z} \]
\[ \tan \varphi = \frac{y}{\sqrt{x^2 + z^2}} \]
\[ x = r \cdot \cos \varphi \cdot \cos \lambda \]
\[ y = r \cdot \sin \varphi \]
\[ z = r \cdot \cos \varphi \cdot \sin \lambda \]
Vector algebra (quick rehearsal of useful formulas)

1. Inner product / dot product of two vectors

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

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

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

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

Used for
- a) calculation of length and distances
- b) angle measurements

Stefan Seipel, Department of Information Technology, Uppsala University
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: \[ P = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}; \quad N = \begin{pmatrix} 2/3 \\ 1/3 \\ 2/3 \end{pmatrix}; \quad |N| = \sqrt{4/9 + 1/9 + 4/9} = \sqrt{9/9} = 1 \]

Case 1: Let \( V = \begin{pmatrix} 10 \\ 7 \\ 15 \end{pmatrix} \)

\[ V' = V - P = \begin{pmatrix} 10 - 1 \\ 7 - 1 \\ 15 - 0 \end{pmatrix} = \begin{pmatrix} 9 \\ 6 \\ 15 \end{pmatrix} \]

\[ |\vec{v}| = N \cdot V' = \frac{2}{3} \cdot 9 + \frac{1}{3} \cdot 6 + \frac{2}{3} \cdot 15 = 6 + 2 + 10 = 18 \]

18>0 -> The polygon is facing towards V.

Case 2: Let \( V = \begin{pmatrix} -5 \\ 7 \\ -9 \end{pmatrix} \)

\[ V' = V - P = \begin{pmatrix} -5 - 1 \\ 7 - 1 \\ -9 - 0 \end{pmatrix} = \begin{pmatrix} -6 \\ 6 \\ -9 \end{pmatrix} \]

\[ |\vec{v}| = N \cdot V' = \frac{2}{3} \cdot -6 + \frac{1}{3} \cdot 6 + \frac{2}{3} \cdot -9 = -4 + 2 - 6 = -8 \]

-8<0 -> The polygon is not facing towards V.
2. Vector product or cross product of two vectors

\[ a \times b = c \]

\[
= \left( a_y b_z - b_y a_z \right) \mathbf{i} + \left( a_z b_x - b_z a_x \right) \mathbf{j} + \left( a_x b_y - b_x a_y \right) \mathbf{k}
\]

\[
= \begin{pmatrix}
  a_y b_z - b_y a_z \\
  a_z b_x - b_z a_x \\
  a_x b_y - b_x a_y
\end{pmatrix}
\]

\( c \) is a vector perpendicular to \( a \) and \( b \).

Used for
a) calculation of surface normal vectors
b) test for co-linearity of two vectors \((c = 0)\)
c) calculation of reference frames
Given: A triangle given by vertices A, B and C

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

\[ A = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}; \quad B = \begin{bmatrix} 2 \\ 1.5 \\ -1 \end{bmatrix}; \quad C = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix} \]

Span vectors:

\[ ab = B - A = \begin{bmatrix} 1 - 2 \\ 1.5 - 1 \\ -1 - 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \\ -1 \end{bmatrix}; \quad ac = C - A = \begin{bmatrix} 1 - 1 \\ 2 - 1 \\ -1 - 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} \]

\[ N = ab \times ac = \begin{vmatrix} 0.5 & 1 \\ -1 & -1 \\ 1 & 0 \end{vmatrix} = \begin{vmatrix} -0.5 - (-1) \\ 0 - (-1) \\ 1 - 0 \end{vmatrix} = \begin{bmatrix} 0.5 \\ 1 \\ 1 \end{bmatrix} \]
Concatenation of Transforms : Matrix Multiplication

Matrix A ; $n \times k$ elements

Matrix B ; $k \times l$ elements

$\Rightarrow C = AB$ defined; and the resulting matrix $C$ has $n \times l$ elements

$\Rightarrow$ generally : $AB$ not equal $BA$ !

$$
c_{ij} = \sum_{s=1}^{k} a_{is} b_{sj}; 1 < i < n, 1 < j < l
$$
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

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

Stefan Seipel, Department of Information Technology, Uppsala University
## Matrix representation of basic geometric transforms

### Scale
\[
S = \begin{pmatrix}
    s_x & 0 & 0 & 0 \\
    0 & s_y & 0 & 0 \\
    0 & 0 & s_z & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]

### Translation
\[
T = \begin{pmatrix}
    1 & 0 & 0 & t_x \\
    0 & 1 & 0 & t_y \\
    0 & 0 & 1 & t_z \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]

### Rotations
\[
R_z(\vartheta) = \begin{pmatrix}
    \cos \vartheta & -\sin \vartheta & 0 & 0 \\
    \sin \vartheta & \cos \vartheta & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]
\[
R_x(\vartheta) = \begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & \cos \vartheta & -\sin \vartheta & 0 \\
    0 & \sin \vartheta & \cos \vartheta & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]
\[
R_y(\vartheta) = \begin{pmatrix}
    \cos \vartheta & 0 & \sin \vartheta & 0 \\
    0 & 1 & 0 & 0 \\
    -\sin \vartheta & 0 & \cos \vartheta & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]
Accumulated/composite transformations

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

Example : $A = R_z T S$

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

Example : $A = R_z T$

Example : $A = T R_z$
Accumulated/composite transformations

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

B)
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

\[ a = \frac{C - A}{|C - A|} \]
\[ b = \frac{(B - A) \times a}{|B - A| \times a} \]
\[ c = \frac{a \times b}{|a \times b|} \]

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

A transformation matrix \( T_{xa} \) rotates the unit vector \((1,0,0,1)\) into the new base vector \(a=(a_x,a_y,a_z,1)\)

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

Stefan Seipel, Department of Information Technology, Uppsala University
Calculation of Reference Frames (Attitude Matrix)

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

Hence:

\[
T_{xa} = \begin{bmatrix} a_x \\ 0 \\ 0 \\ 0 \end{bmatrix}
\]

\[
T_{yb} = \begin{bmatrix}
  t_{11} & t_{12} & t_{12} & t_{14} \\
  t_{21} & t_{22} & t_{23} & t_{24} \\
  t_{31} & t_{32} & t_{33} & t_{34} \\
  t_{41} & t_{42} & t_{43} & t_{44}
\end{bmatrix}, \quad T_{yb} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_x \\ b_y \\ b_z \\ 1 \end{bmatrix}
\]

Hence:

\[
T_{yb} = \begin{bmatrix} ? \\ b_x \\ 0 \\ 0 \end{bmatrix}
\]

\[
T_{zc} = \begin{bmatrix}
  t_{11} & t_{12} & t_{12} & t_{14} \\
  t_{21} & t_{22} & t_{23} & t_{24} \\
  t_{31} & t_{32} & t_{33} & t_{34} \\
  t_{41} & t_{42} & t_{43} & t_{44}
\end{bmatrix}, \quad T_{zc} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} c_x \\ c_y \\ c_z \\ 1 \end{bmatrix}
\]

Hence:

\[
T_{zc} = \begin{bmatrix} 0 \\ 0 \\ c_x \\ 0 \end{bmatrix}
\]

Stefan Seipel, Department of Information Technology, Uppsala University
Calculation of Reference Frames (Attitude Matrix)

Analogously, a transformation matrix $T_{OA}$ translates the origin of the base system $O=(0,0,0,0)$ into the new Origin $A=(A_x, A_y, A_z, 1)$.

$$T_{OA} \cdot O = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} t_{11} \cdot 0 + t_{12} \cdot 0 + t_{13} \cdot 0 + t_{14} \cdot 1 \\ t_{21} \cdot 0 + t_{22} \cdot 0 + t_{23} \cdot 0 + t_{24} \cdot 1 \\ t_{31} \cdot 0 + t_{32} \cdot 0 + t_{33} \cdot 0 + t_{34} \cdot 1 \\ t_{41} \cdot 0 + t_{42} \cdot 0 + t_{43} \cdot 0 + t_{44} \cdot 1 \end{bmatrix} = \begin{bmatrix} A_x \\ A_y \\ A_z \\ 1 \end{bmatrix}$$

Hence: $T_{OA} = \begin{bmatrix} A_x \\ A_y \\ A_z \\ 0 \\ 0 \\ 0 \end{bmatrix}$

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

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

In order to transform a co-ordinate from $O$ into $T$ relative coordinates, $C$ must be transformed with $T^{-1}$.  

Stefan Seipel, Department of Information Technology, Uppsala University
Application of Reference Frames Transforms

The "daily bread-and-butter" in calibrating tracking devices to display devices.

Problem:
Tracking coordinate system $T$ is not co-aligned with display coordinate system $S$.

$h_1$ and $h_2$ are positions measured in reference frame $T$ but are needed in the screen’s local coordinate frame $S$.

Solution:
Register three reference points on-screen that define the origin and base vectors of $S$.

Calculate $S^{-1}$

$h_{1s} = S^{-1} \cdot h_1$
$h_{2s} = S^{-1} \cdot h_2$

Stefan Seipel, Department of Information Technology, Uppsala University
Application of Reference Frames Transform

Calibrating display devices with reference frame in the physical world

Question: For a tracked display and given a given local coordinate within DM, what is the local coordinate within SM?

\[ p_{SM} = SM^{-1} \cdot (DM \cdot p) \]
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?

Explicit mathematical functions of one or more variables

\[ y = f(x, z) \]

Only one dependent co-ordinate

\[ x = p_x(s, t) \]
\[ y = p_y(s, t) \]
\[ z = p_z(s, t) \]

Parametric Surface
How are graphical objects defined?

Finite set of volume points (voxels): Voxelviewer
How are graphical objects defined?

Finite set of surface points: Submarine
3D graphic rendering engines are traditionally optimized for drawing of *graphical primitives*

- 3D points
- 3D line segments
- 3D polygons (mostly triangles)

=> 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 s/w)
The Fixed Function Rendering Pipeline

- Geometric Object
  - Primitives: Points, Lines and Triangles
    - Attributes: Color, Normals, Texture Coordinates
  - Transformation
  - Lighting and Illumination
  - Culling and Clipping
  - Projection
  - Rasterization
  - Interpolation
  - Texel & Pixel Buffer Operations

- Display
Rendering Methods in the Fixed Function Pipeline

• Point Clouds

- Object vertices are drawn as 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
- Very popular for densely sampled surfaces (3D scans)

Examples:
A renaissance of point based rendering

Recent Applications of Point-Based Rendering Techniques

- Avoiding millions of polygons
- Draw surfaces as densely populated point sets
- Increase speed of rendering

- Observe: Point based drawings have complex visual structure
  => good for 3D perception when rendered dynamically
Rendering Methods in the Fixed Function Pipeline

- **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:
Rendering Methods in the Fixed Function Pipeline

• **Solid 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:
Solid 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

Stefan Seipel, Department of Information Technology, Uppsala University
Requirements: Flat Shading vs. Gouraud Shading

**Flat-Shading:**
One surface normal vector per face/polygon.

**Gouraud-Shading:**
Surface normal vectors per vertex. Usually calculated from surrounding polygon normal vectors.

Example: $n_1 = (f_{n1} + f_{n2} + f_{n3} + f_{n4})/4$
Gouraud Shading and Phong Shading

Gouraud

Illumination and color calculation at vertices:

\[ C_1 = (n_1 \cdot l)^\lambda \cdot \xi \]
\[ C_2 = (n_2 \cdot l)^\lambda \cdot \xi \]

Color calculation between vertices by (bi)linear interpolation:

\[ C_n = (1 - \alpha) \cdot C_1 + \alpha \cdot C_2 \quad [0 \leq \alpha \leq 1] \]

More on illumination: see Foley, chapter 16.1

Phong

Illumination and color calculation for each pixel analytical:

For vertex pixels:

\[ C_1 = (n_1 \cdot l)^\lambda \cdot \xi \]
\[ C_2 = (n_2 \cdot l)^\lambda \cdot \xi \]

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

\[ n_s = (1 - \alpha) \cdot n_1 + \alpha \cdot n_2 \quad [0 \leq \alpha \leq 1] \]
\[ c_s = (n_s \cdot l)^\lambda \cdot \xi \]
Differences: Gouraud Shading vs. Phong Shading

Example: \[ I_k = (n_k \cdot l)^\lambda \cdot \xi \]
\[ n_1 = \begin{pmatrix} -1 \\ 4 \\ -1 \end{pmatrix}; n_2 = \begin{pmatrix} 2 \\ 4 \\ 0.5 \end{pmatrix}; l = \begin{pmatrix} -2 \\ -4 \end{pmatrix} \]

\[ \lambda = 1.0; \xi = -1.0 \]

<table>
<thead>
<tr>
<th>n</th>
<th>nx</th>
<th>ny</th>
<th>nx arccos(nl)</th>
<th>l(phong)</th>
<th>l(gouraud)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>-1.00</td>
<td>4.00</td>
<td>-1.00</td>
<td>0.77152</td>
<td>0.771517</td>
</tr>
<tr>
<td>1.00</td>
<td>-0.85</td>
<td>4.00</td>
<td>-0.93</td>
<td>-0.79243</td>
<td>0.722431</td>
</tr>
<tr>
<td>2.00</td>
<td>-0.70</td>
<td>4.00</td>
<td>-1.05</td>
<td>-0.68264</td>
<td>0.812638</td>
</tr>
<tr>
<td>3.00</td>
<td>-0.55</td>
<td>4.00</td>
<td>-1.07</td>
<td>-0.83198</td>
<td>0.831983</td>
</tr>
<tr>
<td>4.00</td>
<td>-0.40</td>
<td>4.00</td>
<td>-1.07</td>
<td>-0.85032</td>
<td>0.850316</td>
</tr>
<tr>
<td>5.00</td>
<td>-0.25</td>
<td>4.00</td>
<td>-1.07</td>
<td>-0.86749</td>
<td>0.867493</td>
</tr>
<tr>
<td>6.00</td>
<td>-0.10</td>
<td>4.00</td>
<td>-1.07</td>
<td>-0.88338</td>
<td>0.883381</td>
</tr>
<tr>
<td>7.00</td>
<td>0.05</td>
<td>4.00</td>
<td>-0.48</td>
<td>-0.89786</td>
<td>0.897862</td>
</tr>
<tr>
<td>8.00</td>
<td>0.20</td>
<td>4.00</td>
<td>-0.40</td>
<td>-0.91084</td>
<td>0.910840</td>
</tr>
<tr>
<td>9.00</td>
<td>0.35</td>
<td>4.00</td>
<td>-0.33</td>
<td>-0.92224</td>
<td>0.922239</td>
</tr>
<tr>
<td>10.00</td>
<td>0.50</td>
<td>4.00</td>
<td>-0.25</td>
<td>-0.93201</td>
<td>0.932007</td>
</tr>
<tr>
<td>11.00</td>
<td>0.65</td>
<td>4.00</td>
<td>-0.18</td>
<td>-0.94012</td>
<td>0.940120</td>
</tr>
<tr>
<td>12.00</td>
<td>0.80</td>
<td>4.00</td>
<td>-0.10</td>
<td>-0.94658</td>
<td>0.946578</td>
</tr>
<tr>
<td>13.00</td>
<td>0.95</td>
<td>4.00</td>
<td>-0.03</td>
<td>-0.95141</td>
<td>0.951406</td>
</tr>
<tr>
<td>14.00</td>
<td>1.10</td>
<td>4.00</td>
<td>-0.05</td>
<td>-0.95465</td>
<td>0.954652</td>
</tr>
<tr>
<td>15.00</td>
<td>1.25</td>
<td>4.00</td>
<td>-0.13</td>
<td>-0.96638</td>
<td>0.966382</td>
</tr>
<tr>
<td>16.00</td>
<td>1.40</td>
<td>4.00</td>
<td>-0.20</td>
<td>-0.96668</td>
<td>0.966681</td>
</tr>
<tr>
<td>17.00</td>
<td>1.55</td>
<td>4.00</td>
<td>-0.28</td>
<td>-0.95664</td>
<td>0.956645</td>
</tr>
<tr>
<td>18.00</td>
<td>1.70</td>
<td>4.00</td>
<td>-0.35</td>
<td>-0.95338</td>
<td>0.953379</td>
</tr>
<tr>
<td>19.00</td>
<td>1.85</td>
<td>4.00</td>
<td>-0.43</td>
<td>-0.949997</td>
<td>0.949997</td>
</tr>
<tr>
<td>20.00</td>
<td>2.00</td>
<td>4.00</td>
<td>-0.50</td>
<td>-0.94561</td>
<td>0.945611</td>
</tr>
</tbody>
</table>

Intensity across span

Stefan Seipel, Department of Information Technology, Uppsala University
How can you achieve better shading than Gouraud?

Using the fixed function pipeline:

Increasing the tessellation
- Increase the numbers of polygons in your object
- Increase the number of normals
- on-screen footprint of polygon equivalent with one pixel

Consequence:
Per-Pixel Evaluation of the Illumination Equations -> Phong Shading

Drawback:
- Vertex processing overhead
- Bandwidth
Object representations in interactive CG

Example: Scene with a teapot

1. Low resolution definition of vertices & polygons rendered using flat shading
2. Combined approach: explicit vertex data and mathematical polynomial functions define curved surfaces, which is retesselated.

Pictures: Copyright (c) 1990 Pixar

Stefan Seipel, Department of Information Technology, Uppsala University
Better solution: Programmable Graphics Hardware

Geometric Object

Primitives: Points, Lines and Triangles
Attributes: Color, Normals, Texture Coordinates

Transformation
Lighting and Illumination
Culling and Clipping
Projection

Rasterization
Interpolation
Texel & Pixel Buffer Operations

Display

Vertex Program
(user definable)

Fragment Program
(user defined)
Vertex- and Fragment Shader: Data Flow

**Vertex Shader:**

Some important accessible attributes (Input)

```glsl
vec4 glPosition
vec3 glNormal
vec4 glColor
vec3 glMultiTexCoord
```

**Output Variables**

```glsl
vec4 gl_Position // must be written to
defloat gl_PointSize // may be written to
defvec gl_ClipVertex // may be written to
```
Vertex- and Fragment Shader : Data Flow

Fragment Shader:

Some important accessible attributes (Input)

```glsl
vec4 gl_FragCoord
```

**user-defined varying variables**

Output Variables

```glsl
vec4 gl_FragColor // must be written to
texture(0, gl_FragCoord)
vec4 gl_FragData[gl_MaxDrawBuffers];
float gl_FragDepth;
```
Example: Programmable Graphics Hardware

A simple shader program for Gouraud Shading

```glsl
#common
varying vec3 color;
varying vec3 normal, light;

#vertex
void main()
{
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    vec3 pos = vec3(gl_ModelViewMatrix * gl_Vertex);
    normal = normalize(gl_NormalMatrix * gl_Normal);
    light = normalize(gl_LightSource[0].position.xyz - pos);
    float diffuse = 0.5;
    float phong = pow(max(0.0, dot(normal, light)), 15.0)*0.5;
    color = vec3(gl_Color.rgb * (phong + diffuse));
}

#fragment
void main()
{
    gl_FragColor = vec4(color, 1.0);
}
```

Stefan Seipel, Department of Information Technology, Uppsala University
A simple shader program for Phong Shading

```
#common
varying vec3 color,pos;
varying vec3 normal,light;

#vertex
void main()
{
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
pos = vec3(gl_ModelViewMatrix * gl_Vertex);
normal = gl_Normal;
light = normalize(gl_LightSource[0].position.xyz - pos);
color = gl_Color.rgb;
}

#fragment
void main()
{
    float diffuse = 0.5;
    float phong = pow(max(0.0, dot(normal,light)), 25.0)*0.5;
    if (normal.x > 0.3) discard;
gl_FragColor = vec4(color * (phong + diffuse), 1.0);
}
```
Programmable Graphics Hardware

Hands-On Demo
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>1 x,y,z,...,u,v</td>
<td>1 4 { 1, 2, 3, 4 } { ........ }</td>
</tr>
<tr>
<td>2 x,y,z,...,u,v</td>
<td>2 3 { 6, 4, 3 } { ........ }</td>
</tr>
<tr>
<td>3 x,y,z,...,u,v</td>
<td>3 3 { 5, 4, 6 } { ........ }</td>
</tr>
<tr>
<td>n x,y,z,...,u,v</td>
<td>4 4 { 3, 2, 7, 6 } { ........ }</td>
</tr>
</tbody>
</table>

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

**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 hierarchical order
Simple Robot: Modeling Hierarchy

Tree of involved instances

List of generic objects used
Creating the Model Hierarchy

Base
  ↓
Trunk
  ↓
Head
  ↓
Left_Arm
  ↓
Right_Arm

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);
Simple Robot: The Scene Graph

Start robot.exe

“base”

“trunk”

“head”

“left_arm”

“right_arm”

Stefan Seipel, Department of Information Technology, Uppsala University