Introduction to Computer Graphics

Marie-Paule Cani : Cours
Estelle Duveau : CTD OpenGL
• **3D animation**
  – « Disney Effects »
  (Luxo Jr (1986)
  Pixar animation studios
  Director: John Lasseter)
Computer Graphics

- Special effects
  - Seam-less mix of real & virtual
Computer Graphics

- Games
  - Immersion through interaction
Computer Graphics

• Simulation : « serious games »
  – Predictability & interaction
Computer Graphics

- Computer Aided Design (CAD)
  - Virtual prototypes
Computer Graphics

• Architecture
  – Real-time exploration
Computer Graphics

• Virtual reality
  – Multi-sensorial immersion
• Augmented reality
Computer Graphics

• Visualization
  - Visual exploration of results, interaction
Computer Graphics

- Medical imaging
  - Understanding, planning, on-line monitoring
Computer Graphics

- Design
- 3D animation
- Special effects
- Games
- Simulators
- Visualization

Tools for artists
Realism
Real-time

Computer Graphics Research
What you will learn

- Overview of Computer Graphics (including vocabulary)
  - Modeling: create 3D geometry
  - Animation: move & deform
  - Rendering: 3D scene → image
- How basic techniques work
- Practice with OpenGL (C++)
- Introduction to research: case studies
  - Choose/combine/extend existing techniques to solve a problem
What you will not learn

- Advanced techniques in detail
- Programming the Graphics Hardware (GPU)
- Artistic skills
- Game design
- Software packages
  (CAD-CAM, 3D Studio Max, Maya, Photoshop, etc)

Following up: MOSIG M2 “GVR” & ENSIMAG “IRV”
Text books

- No book required
- References
  - *3D Computer Graphics*
    Alan H. Watt
Course schedule (3h a week, A009 or ARV)
Marie-Paule Cani & Estelle Duveau

04/02 Introduction + Projective rendering: graphics pipeline, shading
11/02 Parametric modeling: representations + design tools
25/02 Introduction to OpenGL: C + TD
04/03 Implicit surfaces 1 + CTD matrices & hierarchies
11/03 Implicit surfaces 2 + C OpenGL lighting, materials
18/03 Textures, aliasing + TD OpenGL lighting, materials
25/03 Textures in OpenGL: C + TD
01/04 Procedural & kinematic animation + TD procedural anim
08/04 Physics: particle systems + TD physics 1
22/04 Physics: collisions, control + TD physics 2
29/04 Animating complex objects + Realistic rendering
06/05 Talks: results of cases studies
Basic, real-time display?

Projective rendering

Done by the graphics hardware via OpenGL or DirectX

- **Input:** Scene
  - 3D models (Faces & normals)

- **Goal**
  - Image from camera
    Made of pixels
Basic, real-time display?
Projective rendering

2 ingredients:

Graphics pipeline
From a 3D scene to a 2D image
• based on geometry

Local illumination
Which color in each pixel?
• based on optics
1. Create 3D models
   • in local frames, faces = vertices + normals

2. Build the scene
   • place instances of models in the “world frame”
   • add materials, virtual lights, and a camera
**Representation of transformations**

- From frame to frame (rotate, translate, scale)?
  - Transformations represented by 4x4 matrices

\[
\begin{pmatrix}
    x' \\
    y' \\
    z' \\
    w'
\end{pmatrix} = \begin{pmatrix}
    a & b & c & d \\
    e & f & g & h \\
    i & j & k & l \\
    m & n & o & p
\end{pmatrix} \begin{pmatrix}
    x \\
    y \\
    z \\
    w
\end{pmatrix}
\]

\[p' = M p\]
Why 4x4? Homogeneous coordinates

- $w$ will be used for projective transformations
- Cartesian coordinates: $w = 1$
- From projective to cartesian: divide by $w$

\[
\begin{pmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{pmatrix} =
\begin{pmatrix}
    a & b & c & d & 0 \\
    e & f & g & h & 0 \\
    i & j & k & l & 0 \\
    0 & 0 & 0 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
    x \\
    y \\
    z \\
    1
\end{pmatrix}
\]

Affine transformation
Affine transformations

Translation

\[
T = \begin{bmatrix}
1 & 0 & 0 & T_x \\
0 & 1 & 0 & T_y \\
0 & 0 & 1 & T_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Rotation: Euler angles

\[
\begin{align*}
R_x &= \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \psi & -\sin \psi & 0 \\
0 & \sin \psi & \cos \psi & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \\
R_y &= \begin{bmatrix}
\cos \phi & 0 & \sin \phi & 0 \\
0 & 1 & 0 & 0 \\
-\sin \phi & 0 & \cos \phi & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \\
R_z &= \begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
R = R_z \cdot R_y \cdot R_x
\]
Composition of transformations

Multiplication of matrices: \( p' = T(Sp) = TS\ p \)

\[
TS = \begin{bmatrix}
1 & 0 & 3 \\
0 & 1 & 1 \\
0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
= \begin{bmatrix}
2 & 0 & 3 \\
0 & 2 & 1 \\
0 & 0 & 1 \\
\end{bmatrix}
\]
Scale then translate: $p' = T(Sp) = TS\ p$

Translate, then scale: $p' = S(Tp) = ST\ p$

Not commutative !!!
Graphics pipe-line

3. Convert the scene to the camera frame
   – « cull » the faces that look in the opposite direction

Normal ≈ vector to the caméra ?
4. Convert to the screen frame (projective transformation!)
   - The viewing frustrum becomes a parallelogram
   - «clipping» operations to
     • suppress faces outside the frustrum, cut intersecting ones
Perspective projection to image plane?

- Project all points to the $z = d$ plane, eyepoint at the origin.

\[
\begin{align*}
x_p &= \frac{d \cdot x}{z} = \frac{x}{z/d} \\
y_p &= \frac{d \cdot y}{z} = \frac{y}{z/d} \\
z_p &= d 
\end{align*}
\]

\[
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix} =
\begin{pmatrix}
x_p \\
y_p \\
z_p \\
1
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
\]

homogenize
Graphics pipe-line

5. Compute the image
- Rasterize each face into pixels (x,y)
- Suppress hidden parts
- Compute a color for each pixel
Rasterize faces into pixels?

- Primitives are continuous; screen is discrete
  - triangles are described by a discrete set of vertices
  - but they describe a continuous area on screen
Rasterize faces into pixels?

- Scan Conversion: approximation into pixels
  - Check pixels in BB wrt the 3 line equations
  - Scanline rasterization: increment from corner vertices
Graphics pipeline

Remove the hidden parts of each triangle?
Else the last one will appear « above »
Remove hidden parts of each triangle?
  – First method: the painter’s algorithm
    • Sort the faces
    • Display triangles starting with the farthest

  • Cost \( n(\log n) \)
  • Problems!
Graphics pipeline

Remove hidden parts?

- Use a « Z-buffer » (available thanks to memory)
  - A second array, as large as the image
  - Stores the current z value at each pixel
    (the associated color being in the image buffer)

Algo

- Init with all pixel at max distance
- For each face, for each pixel P
  - update color and z-value iff \((z < \text{current z-value}(P))\)
Graphics pipeline

• Which color should be displayed?
  – Uniform colors would not work!
  – Given by a « local illumination » model
Local illumination

Which color shall we display in each pixel?
⇒ Depends on the local amount of light coming back to the eyes
⇒ So it depends on:
  – where the surface element is in 3D
  – its orientation w.r.t. lights & camera
  – the material the surface is made of
Phong’s local illumination

- A constant « ambiant » term
- Direct lighting from the sources
  - no shadows
- Opaque objects only

diffuse

specular
Phong’s local illumination

\[ I = K_a + \sum I_s (K_d L \cdot N + K_s (R \cdot V)^n) \]

ambient    diffuse    specular
Phong’s local illumination

ks

n
Direct application

- A single normal by face
- Uniform colors!
Gouraud’s shading

- A normal by face
- Illumination on each vertex
- Bi-linear interpolation

Better!

Some reflexions can be missed
Phong’s shading

- A normal by vertex
- Interpolate normal directions
- Illumination at each pixel

Correct!

Still missing:
- Cast shadows
- Extended light sources
- Transparency

Specular reflection
Phong’s shading