CSE528 Computer Graphics: Theory, Algorithms, and Applications

Hong Qin
Department of Computer Science
State University of New York at Stony Brook (Stony Brook University)
Stony Brook, New York 11794--4400
Tel: (631)632-8450; Fax: (631)632-8334
qin@cs.sunysb.edu
http://www.cs.sunysb.edu/~qin
Key Components

• **Introduction**
  – Overview, definition
  – Various application examples and areas
  – Graphics history
  – Graphics software and hardware systems
  – Graphics programming
  – User-computer interface, special input/output devices
Key Components

• **Geometry and Mathematics**
  – Curves, and surfaces
  – Solid geometry and volumetric models for datasets
  – 3D geometric transformation
  – Data structures
Key Components

• **Geometric Modeling**
  - Curves
  - Surfaces
  - Shape modeling, matching, analysis
  - Shape parameterization
  - Solid modeling
Key Components

• Rendering
  – Object hierarchies
  – Ray tracing
  – Object and image order rendering
  – Graphics rendering pipeline
  – Color perception and color models
  – Basic optics
  – Visibility
Key Components

• **Image-based techniques**
  - Sampling
  - Filtering
  - Anti-aliasing
  - Image analysis and manipulation
Key Components

• Advanced Topics
  – Animation
  – Transparency and shadows
  – Texture mapping
  – Image-based rendering and modeling
  – Advanced modeling techniques
  – Case studies
  – Software packages
  – ............
Computer Graphics Pipeline
What is Computer Graphics?

- Computational process of generating images from models and/or datasets using computers
- This is called *rendering* (*computer graphics was traditionally considered as a rendering method*)
- A rendering algorithm converts a geometric model and/or dataset into a picture
What is Computer Graphics?

This process is also called *scan conversion* or *rasterization*.
Computer Graphics

• **Computer graphics consists of:**
  1. Modeling (representations)
  2. Rendering (display)
  3. Interaction (user interfaces)
  4. Animation (combination of 1-3)

• Usually “computer graphics” refers to rendering
A Classical Classification

Image Processing

2D Images

Computer Graphics

3D Models

Computer Vision

Computer Graphics

3D Models

2D Images

Image Processing
Surface Graphics

• Surface representations are good and sufficient for objects that have *homogeneous* material distributions and/or are not *translucent* or *transparent*

• Such representations are good only when object boundaries are important (in fact, only boundary geometric information is available)

• Examples: furniture, mechanical objects, plant life

• Applications: video games, virtual reality, computer-aided design
Applications
Computer Animation
Computer Aided Design (CAD)
Architecture
Video Games
Digital Art
Surface Graphics – Pros and Cons

- **Good**: explicit distinction between inside and outside makes rendering calculations easy and efficient
- **Good**: hardware implementations are inexpensive
- **Good**: can use tricks like texture mapping to improve realism
- **Bad**: an approximation of reality
- **Bad**: does not let users peer into and through objects
Volume Graphics

- Surface graphics doesn’t work so well for clouds, fog, gas, water, smoke and other *amorphous phenomena*
- “amorphous” = “without shape”
- Surface graphics won’t help users if we want to explore objects with very complex internal structures
- Volume graphics provides a good technical solution to these shortcomings of surface graphics
- Volume graphics includes *volume modeling (representations)* and *volume rendering* algorithms to display such representations
Scientific Visualization
Visualization (Isosurfaces)
Visualization (Volume Rendering)
Computer Simulation
Graphics Pipeline

Modeling → Rendering
Computer Graphics Pipeline

1. Model
2. Geometry Processing
3. Projection
4. Rasterization
5. Display
Visualization Pipeline

- Computational Methods
  - finite element
  - finite difference
  - boundary element
  - numerical analysis

- Measured Data
  - CT, MRI, ultrasound
  - satellite
  - laser digitizer
  - stocks/financial

Data

Transform → Map → Display
Relevant Disciplines

data

sensors, scanners, cameras
sampling/scanning

geometric model (structures)
polygonization

discretization

computer graphics
computer vision

image (signal)

display device

film recorder

computation/simulation

super-computers

image processing
Lights, Cameras and Objects

• In reality, how are we able to see things in the real world?
• What’s the computational process that occurs?
• Let us start this process:
  1. Open eyes
  2. Photons from light source strike object
  3. Bounce off object and enter eye
  4. Brain interprets image you see

We will have to simulate this process computationally!
Lights, Cameras and Objects

• Rays of light emitted by light source
• Some light strikes object we are viewing
  – Some light absorbed
  – Rest is reflected
  – Some reflected light enters into our eyes
Lights, Cameras and Objects

• How do we simulate light transport in a computer?
• Several ways
• *Ray-tracing* is one
• Start at eye and trace rays the scene
• If ray strikes object, bounces, hits light source → we see something at that pixel
• Most computer applications don’t use it. Why?
• With many objects very computationally expensive
Surface Ray-Tracing
Rendering Processes: Image-Order and Object-Order

- Ray-tracing is an *image-order* process: operates on *per-pixel basis*
- Determine for each ray which objects and light sources ray intersects
- Stop when all pixels processed
- Once all rays are processed, final image is complete
- *Object-order* rendering algorithm determines for each object in scene how that object affects final image
- Stop when all objects processed
Rendering Processes: Image-Order and Object-Order

- **Image-order approach:** start at upper left corner of picture and draw a dot of appropriate color
- **Repeat for all pixels** in a left-to-right, top-to-bottom manner
- **Object-order approach:** paint the sky, ground, trees, barn, etc. back-to-front order, or front-to-back
- **Image-order:** very strict order in which we place pigment
- **Object-order:** we jump around from one part of the regions to another
Rendering Processes: Image-Order and Object-Order

- Advantages and disadvantages of each
- Ray-tracing can produce very realistic looking images, but is very computationally expensive
- Object-order algorithms more popular because hardware implementations of them exist
- Not as realistic as ray-tracing
Surface Rendering

- We have considered interaction between light rays and object boundaries
- This is called *surface rendering* and is part of *surface graphics*
- Computations take place on boundaries of objects
- Surface graphics employs *surface rendering* to generate images of surface’s mathematical and geometric representations
Mathematical Surfaces (Sphere)

• Equation of a sphere: \(x^2 + y^2 + z^2 = r^2\)
• How thick is the surface?
• Are there objects in real world thickness zero?
Surface Graphics

• Can you think of objects or phenomena for which this approach to rendering will fail?
• When is a surface representation not good enough?
• Would a surface representation suffice to represent the internal structure of the human body?
From Surface Graphics to Volume Visualization

- **Visualization**: transformation of data into graphical form
- **Object-oriented-based approach**: data are the objects, transformations are the methods
Data Visualization Example

• Usually we evaluate the equation of a sphere for a particular radius, \( r \):

\[
x^2 + y^2 + z^2 = r^2
\]

• Suppose we evaluate it for different values of \( r \)?

• We get a solid sphere

• Now imagine we evaluate it for any value of \( x, y, z \) and \( r \)

\[
F(x, y, z) = x^2 + y^2 + z^2 - r^2
\]

• We get what’s called a field function

• You plug in some values for \( x, y, z, r \) and get some number. That number is “located” at position \((x, y, z)\)
Data/Model Visualization Example

- **A quadric** is a special function with maximum degree 2:

\[
F(x, y, z) = a_0 x^2 + a_1 y^2 + a_2 z^2 + a_3 xy + a_4 yz + a_5 xz + a_6 x + a_7 y + a_8 z + a_9
\]

- **A solid sphere** is an example of a quadric with \(a_3, a_4, a_5, a_6, a_7,\) and \(a_8\) all equal to zero.

- If those values aren’t zero, we get some pretty strange shapes.

- Imagine squishing a **solid** rubber ball (i.e., not a hollow ball, like a tennis ball).
Data Visualization Example

- If we plug in $x, y, z, r$ for any quadric, we can get some very strange-looking field functions. Here's an example:

$$F(x, y, z) = a_0 x^2 + a_1 y^2 + a_2 z^2 + a_3 xy + a_4 yz + a_5 xz + a_6 x + a_7 y + a_8 z + a_9$$
Volumetric Representations

- A volumetric data-set is a 3D regular grid, or 3D raster, of numbers that we map to a gray scale or gray level.
- Where else have you heard the term raster?
- An 8-bit volume could represent 256 values [0, 255].
- Typically volumes are at least $200^3$ in size, usually larger.
- How much storage is needed for an 8-bit, $256^3$ volume?
Volume Graphics

- Volumetric objects have interiors that are important to the rendering process (what does that mean?)
- Interior affects final image
- Imagine that our rays now don’t merely bounce off objects, but now can penetrate and pass through
- This is known as *volumetric ray-casting* and works in a similar manner to surface ray-tracing
Volumetric Ray-Tracing
Volume Rendering

• In volume rendering, imaginary rays are passed through a 3D object that has been discretized (e.g., via CT or MRI)

• As these viewing rays travel through the data, they take into account of the intensity or density of each datum, and each ray keeps an accumulated value
Volume Rendering

• As the rays leave the data, they comprise a sheet of accumulated values

• These values represent the volumetric data \textit{projected} onto a two-dimensional image (the screen)

• Special mapping functions convert the grayscale values from the CT/MRI into color
Volume Rendering

- Semi-transparent rendering
Volume Graphics

- Good: maintains a representation that is close to the underlying fully-3D object (but discrete)
- Good: can achieve a level of realism (and “hyper-realism”) that is unmatched by surface graphics
- Good: allows easy and natural exploration of volumetric datasets
- Bad: extremely computationally expensive!
- Bad: hardware acceleration is very costly ($3000+ vs $200+ for surface rendering)
Surface Graphics vs. Volume Graphics

- Suppose we wish to animate a cartoon character on the screen
- Should we use surface rendering or volume rendering?
- Suppose we want to visualize the inside of a person’s body?
- Now what should approach we use? Why?
- Could we use the other approach as well? How?
- We could visualize body as collection of surfaces
Graphics Hardware
Virtual Reality Systems
Virtual Reality Systems

[Image of a person using a VR headset and controllers]
Virtual Reality Systems
Trackball, Joystick, Touch Pad
Haptics Device (Phantom 1.0)
3D Laser Range Scanner
3D Laser Range Scanner
3D Camera
Digital Fringe Projector
OpenGL

- Most widely used 3D graphics Application Program Interface (API).
- Truly open, independent of system platforms.
- Reliable, easy to use and well-documented.
- Default language is C/C++.
OpenGL

- The **GL** library is the core OpenGL system:
  - modeling, viewing, lighting, clipping

- The **GLU** library (GL Utility) simplifies common tasks:
  - creation of common objects (e.g. spheres, quadrics)
  - specification of standard views (e.g. perspective, orthographic)

- The **GLUT** library (GL Utility Toolkit) provides the interface with the window system.
  - window management, menus, mouse interaction
OpenGL

• To create a red polygon with 4 vertices:

```c
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_POLYGON);
   glVertex3f(0.0, 0.0, 3.0);
   glVertex3f(1.0, 0.0, 3.0);
   glVertex3f(1.0, 1.0, 3.0);
   glVertex3f(0.0, 1.0, 3.0);
glEnd();
```

• `glBegin` defines a geometric primitive:

   ```c
   GL_POINTS, GL_LINES, GL_LINE_LOOP, GL_TRIANGLES, GL_QUADS, GL_POLYGON...
   ```

• All vertices are 3D and defined using `glVertex`
FLTK

- Fast Light Tool Kit (FLTK)
- www.fltk.org
- C++ oriented
  - A set of UI classes such as Window, box, etc.
- Can mix use with GLUT
- FLUID: fast light UI Designer
  - Fast creation of GUI
  - Automatically writes parts of GUI code from a graphical spec
  - Good for elaborate interfaces
Comments on Programming

• **OpenGL, VTK, plus Glui**
  – Simple, easy to program, limitations

• **OpenGL, VTK, plus FLTK**
  – Cross platform, more powerful

• **OpenGL, VTK, plus Visual C++**
  – Super!
  – Only run under windows system