Migrating to an Object-Oriented Graphics API

Course Objective

An introduction to object-oriented graphics APIs that is more fundamental than any one specific API.

Course Presenters

- David Blythe
- Sherif Ghali
- Lutz Kettner
- Henry Sowizral

Course Audience

- Familiar with an OOP
- Migrating to an object-oriented API
- Building an API

Material Addressed

- Introduction to current OO APIs
- Common and different API features
- Design your own mini scene graph API

Material Not Addressed

- Not an objective:
  - Details sufficient for programming
  - Suggest that any one API is “better”
- Corollary:
  - Tutorial/reference manuals are needed
  - API selection
O-O vs. Scene Graph

<table>
<thead>
<tr>
<th></th>
<th>Non O-O</th>
<th>O-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non Scene Graph</td>
<td>OpenGL</td>
<td>C++/Java binding of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OpenGL</td>
</tr>
<tr>
<td>Scene Graph</td>
<td>PHIGS</td>
<td>Open Inventor / Java3D / Fahrenheit/XSG</td>
</tr>
</tbody>
</table>

Application Programmer Interface

- Classes and methods defined by a library

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Client code</td>
</tr>
<tr>
<td>Scene graph library</td>
</tr>
<tr>
<td>Low-level library</td>
</tr>
</tbody>
</table>

Course Outline

- Real and Projective Geometry
- Scene Graphs
- Lighting and Visibility
- Input and User Interaction
- New Features of Scene Graph APIs

Affine and Projective Geometry

- Designing Geometry Classes (Ghali)
- Classes for Affine and Projective Geometry (Blythe)
- Geometry Class Design Issues (Sowizral)
- Robustness of Geometry Classes (Kettner)

Designing Geometry Classes

Sherif Ghali
Max Planck Institute
Why Talk About Geometry Classes?

Does the design of elementary geometry classes warrant a lengthy discussion?

Issues Addressed

- Introduction and motivation for robustness
- Options in designing geometry classes (DB)
- A recent design of geometry classes (HS)
- A recent design of a robust library (LK)

Introduction and Motivation for Robustness

Is point A inside polygon P?

Do segments L and M intersect?

where?

Does ray R intersect polygon Q?

where?

Precision in 3D: Binary Space Partitioning

Partitioning plane

Handling Precision Problems

- Use an epsilon
- Use exact arithmetic
Handling Precision Problems

- Use an epsilon
- Use exact arithmetic

Use an Epsilon

- Introduce a line:
  \[
  \text{if ( d > -\ Epsilon \ || \ d < \ Epsilon )}
  \]
  \[
  d = 0;
  \]

Segment Intersection

Segment Coincidence

point ab: intersection of A and B
point ac: intersection of A and C

Is ( ab = ac ) true?
Using Floating Point Numbers

```cpp
#include <LEDA/point.h>
#include <LEDA/segment.h>

main() {
    segment A ( point( 13, 7 ), point( 1, 3 ) );
    segment B ( point( 8, 8 ), point( 6, 2 ) );
    segment C ( point( 1, 8 ), point( 13, 2 ) );
    point ab, ac;
    A.intersection ( B, ab );
    A.intersection ( C, ac );
    std::cout << ab << std::endl << ac << std::endl;
    assert ( ab == ac );
}
```

Using Floating Point Numbers - Output

```
(7.0000000000000009,5)
(7,5)
Assertion failed: ab == ac
```

Using Rational Numbers

```cpp
#include <LEDA/rat_point.h>
#include <LEDA/rat_segment.h>

main() {
    rat_segment A ( rat_point( 13, 7 ), rat_point( 1, 3 ) );
    rat_segment B ( rat_point( 8, 8 ), rat_point( 6, 2 ) );
    rat_segment C ( rat_point( 1, 8 ), rat_point( 13, 2 ) );
    rat_point ab, ac;
    A.intersection ( B, ab );
    A.intersection ( C, ac );
    std::cout << ab << std::endl << ac << std::endl;
    assert ( ab == ac );
}
```

Using Rational Numbers - Output

```
(7,5)
(7,5)
```

Kernel-independent Code

- Use either
  ```cpp
  #define POINT point
  or
  #define POINT rat_point
  ```

Kernel-independent Code

- Using floating point numbers:
  ```cpp
  #include <LEDA/point.h>
  #include <LEDA/segment.h>
  #include <LEDA/float_kernel_names.h>
  ```
- Using rational numbers:
  ```cpp
  #include <LEDA/rat_point.h>
  #include <LEDA/rat_segment.h>
  #include <LEDA/rat_kernel_names.h>
  ```
Kernel-independent Code

main() {
    SEGMENT A ( POINT( 13, 7 ), POINT( 1, 3 ) );
    SEGMENT B ( POINT( 8, 8 ), POINT( 6, 2 ) );
    SEGMENT C ( POINT( 1, 8 ), POINT( 13, 2 ) );
    POINT ab, ac;
    A.intersection ( B, ab );
    A.intersection ( C, ac );
    std::cout << ab << std::endl << ac << std::endl;
    assert ( ab == ac );
}

LEDA

- Library of Efficient Data structures and Algorithms
  - Floating-point kernel
    - point, segment, ...
  - Rational kernel
    - rat_point, rat_segment, ...

Methods of Changing Kernels

- By using aliasing
  - LEDA
- By using genericity
  - CGAL

Classes for Affine and Projective Geometry

- What is your objective?

Designing Classes for Affine and Projective Geometry

David Blythe
RouteFree
Classes for Affine and Projective Geometry

- **Basic Classes**
  - Tuple (3 component, 4 component color)
  - Euclidean Points (2D, 3D)
  - Homogenous Points (2D, 3D)
  - Vectors (2D, 3D)
  - Normal Vectors (2D, 3D)
  - Affine Transforms (3x3 Matrix, 4x4, 4x3, …)
  - Frustum, Polytope, Ray, LineSegment

- **Type Explosions**
  - #Types X #Dimensions X #Data representation
    - e.g., integer 2D point, float 2D point, double 3D point
    - integer 3D point, float 3D point, double 3D point
  - Why all the types?
    - 2D is more convenient and saves space over 3D
    - Double precision is often necessary but takes 2x the space

- **Implementation Tips**
  ```
  struct Point3 {
      float v[3];
  };
  public member
  obvious memory layout
  generic access
  ```

Classes for Affine and Projective Geometry

- **Type Explosions**
  - Strong Typing?
    - Good for encouraging correctness
      - Don’t add points \((p_1 + p_2)/2\)
      - Treat normal vectors properly \(\mathbf{n} \cdot \mathbf{M}^{-1} \mathbf{n}\)
    - Source of implementation complexity
      - Multiple methods with similar behavior
      - Limited inheritance limited reuse
      - tuple + point (wrong if +/- defined on tuple)

- **Solutions**
  - Templates
    - Requires careful design (generic programming)
    - May end up forcing instantiations anyway
    - Performance limitations?
    - Limit number of types supported
      - Engineering compromise
      - Dependent on application domain

- **Compromises**
  - Compress Euclidean Point and Homogenous Point?
    - What about \(w\)?
  - Compress Normal Vector and Vector?
  - Compress Tuple and Point?
  - Allow more inheritance?
Classes for Affine and Projective Geometry

- IRIS Performer/Open Inventor
  - No Strong Typing
  - Everything a tuple (Vec*)
- Java3D
  - More strongly typed, fewer basic types
    - double vs. float vs. int
  - Inheritance model breaks typing
    - add/sub defined on Tuple, inherited in Point

Classes for Affine and Projective Geometry

- Naming Issues
  - Encode
    - basic type (float, double, int, short),
    - dimensionality (2D, 3D, ...)
    - intent (point, vector, ...)
  - {Tuple, Point, Vector}{1,2,3,4}{l,f,d,s}
    - $3 \times 4 \times 4 = 48$ classes

Classes for Affine and Projective Geometry

- Operator Support
  - Point - Point, Point*scalar
  - Point +/- Vector, Vector*scalar
  - Vector +/- Vector, cross, dot, normalize, length, length$^2$
  - Transform Point, Vector
  - Build Transform (Scale, Rotate, Translate, Invert)

Classes for Affine and Projective Geometry

- Operator Explosions
  - Allow mixtures of representations?
    - e.g., Vector3f*Matrix3d
  - Ignore homogeneous term sometimes
    - e.g., $M_p$, is point always translated?
  - Seldom used operators
    - tensor product, ...
  - Performance tuned special cases

Classes for Affine and Projective Geometry

- Row Vectors versus Column Vectors
  - Computer graphics literature mixed (up :)
    - Foley & van Dam switched to column vectors
  - What does C/C++ 2D array map to?
    - Low-level APIs 2D arrays map to row vectors
  - OpenGL, DirectX
    - Efficiency problems if OO-API mismatches LL-API
  - Should use column vectors (ARB_transpose_matrix)

Classes for Affine and Projective Geometry

- Representing Rotations
  - Matrix (3x3)
  - Axis/Angle ($\theta$,x,y,z)
  - Euler Angles (roll, pitch, yaw)
  - Quaternions
Classes for Affine and Projective Geometry

- **Bounding Volumes**
  - Bounding boxes (axis aligned)
  - Tight bounds
  - Bounding spheres
  - Low cost
  - Bounding ellipsoids
  - Oriented bounding boxes, ...
    - Compromise between box/sphere

- **Bounding Volume Operations**
  - void create(Point points[]);
  - void create(Bound bounds[]);
  - void extend(Point p);
  - void extend(Bound b);
  - bool overlaps(Bound b);

Classes for Affine and Projective Geometry

- **How Far Should Toolkit Go?**
  - Toolkit versus application framework
  - Provide comprehensive math library?
    - Larger than 4x4 matrices, SVD, linear solvers, ...
  - Provide what typical 3D applications need?
    - Point, Vector, Normal, Transform, ...
    - What about CAD applications, collision detection, ...
  - Match the application domain

- **How Far Should A Toolkit Go?**
  - Provide a basis-set that can be extended or aliased with another library
    - Points, vectors, transforms don’t contain extra state (no flags)
    - Obvious memory layout
    - Allow further derivation

Geometry Class Design Issues

Henry A. Sowizral
Sun Microsystems, Inc.
henry.sowizral@sun.com

Copyright © 2000 Henry A. Sowizral

Geometry Concerns

- Matching representation to needs
  - Programmer (ease of use)
  - Underlying hardware (speed)
- Memory usage
Optimization Points

- Programming
- Rendering
- Manipulating
- Modeling

Geometry Class Requirements

- Expressiveness
- Representational structure
  - Triangles
  - Analytical
- Efficiency
  - Memory, Disk, and Bandwidth (communication)
  - Manipulation (editing)
  - Rendering

Expressiveness

- Rendering – tessellated object geometry
- Manipulation
  - Simplification – topology and geometry
  - Editing – facet and other structural information
- Access to non-geometric information
  - Scene placement
  - Material properties
  - Business information

Basic Representations

- Basic geometry
  - Points
  - Lines
  - Triangles (Quads)
- Stripped geometry
  - Polylines
  - Triangle fans / Triangle strips
- Indexed geometry

Basic Geometry

- Points
- Lines

Basic Geometry, cont.

- Triangles
- Quads (not so basic)
Stripped Geometry

- Triangle strips
- Line strips
- Triangle fans

Indexed Geometry

- Vertex information
  - Basic vertex information
  - Stored in array
  - Typically not duplicated
- Array(s) of indices
  - An index value references the corresponding vertex information
  - Index order specifies the order of vertex information

Analytic Representations

- Surfaces
  - NURBS
    - Trim curves
  - Subdivision surfaces
    - Interpolating
    - Approximating
  - Analytic surfaces

Graphics Outpacing CPU

Per Vertex Information

- Typical triangle: 24-48 bytes
  - Lighting normals (12 bytes = 3 x 4 bytes)
  - Colors (12 | 16 bytes = 3 | 4 x 4 bytes)
  - Coordinates (12 bytes = 3 x 4 bytes)
  - Texture coordinate(s) (k x 12 bytes = k x 3 x 4 bytes)

Speeds Too Slow

- 10 Million triangles = 240-480 Mbytes/second
- 12-24 Million instructions just for transfer
  - Wait states
  - Indexing
  - Any additional computation
- Minimum bus speed
  - 2 trips on the bus
  - 480-960 Mbytes/second on main bus
  - 240-480 on main bus
  - 240-480 on ancillary bus
Efficiency

- Explicit Versus Analytic Representations
  - Pre-tessellated
    - Known structure(s)
    - Can match to underlying hardware — very efficient
  - Non-tessellated: tessellate
    - Each time (interactive LOD)
    - Once and cache
    - When needed and cache

Compression Techniques

- Fewer bits per vertex, per triangle
  - Binning: 433, etc.
- Indexed representations
- Connectivity (implicit and explicit)
- Higher compression
  - Binning
  - Run length encoding
  - Huffman encoding

Flexibility Versus Simplicity

Versus Completeness

- What is your favorite format?
  - Analysis of use results in best representation
- What do you want to optimize for?
  - Memory usage
  - Rendering speed
  - Ease in manipulation
  - Ease in modeling

Modeling Versus Rendering

- Rendering
- Modeling

Memory Usage

- What is the nature of the contract?
- Are internal copies required?

Indexed Versus Non-indexed

- Memory
  - Does it really save memory?
- Speed
  - Can it impact speed?
Shared Versus Non-Shared

• Sharing is it good or bad?
  • Under the cover costs
  • To duplicating or not to duplicate

What About Data Staging?

Robustness of Geometry Classes

Lutz Kettner
UNC Chapel Hill
kettner@cs.unc.edu

Robustness in Geometry

• Robustness in Geometry
• Templates in C++
• Generic Programming
• Geometry Classes in CGAL

Robustness in Geometry

• Rounding Errors
• Degeneracies
Rounding Errors

```c
#include <assert.h>

double det(double a, double b, double c, double d) {
    return a * d - b * c;
}

int main() {
    double px = 1.0, py = 0.0;
    double qx = 1.3, qy = 1.7;
    double rx = 2.2, ry = 6.8;
    // test for collinearity
    assert(det(px-rx, qx-rx, py-ry, qy-ry) == 0.0);
}
```

Rounding Errors (2)

Base 10, mantissa 2, rounding to zero

\[
y = 4.3 x / 8.3 \\
y = 1.4 x / 2.7
\]

Degeneracies

- Duplicate points (set versus sequence)
- Three points on a line
- Four points on a circle

Degeneracies (2)

- Explicit handling of special cases
  - sometimes easy (segment intersection in the plane)
  - sometimes a combinatorial explosion
- Symbolic perturbation
  - exact arithmetic, validity of solution, postprocessing
- Random perturbation
  - validity check and restart for Las Vegas type

Rounding Errors (3)

- Redesign algorithms for FP arithmetic
  - works for some problems, no general theory
- Exact arithmetic
  - long integers, rationals, reals (incl. k-th roots)
- FP filter for efficiency
  - error bounds: static, semi-static, dynamic
- Type of arithmetic and filter depends on the application
  - Flexibility
Special Case Handling

Binary Decision Tree
- \(x < 0\)
- \(x > 0\)

Trinary Decision Tree
- \(x < 0\)
- \(x > 0\)
- \(x = 0\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x,y)\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x,y + \text{eps})\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x + \text{eps}^2, y + \text{eps})\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x,y)\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x,y + \text{eps})\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x + \text{eps}^2, y + \text{eps})\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x,y)\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x,y + \text{eps})\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x + \text{eps}^2, y + \text{eps})\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x,y)\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x,y + \text{eps})\)

Symbolic Perturbation

Point in Polygon

Horizontal ray shooting

\(P = (x + \text{eps}^2, y + \text{eps})\)

Robustness of Geometry Classes

- Robustness in Geometry
- Templates in C++
- Generic Programming
- Geometry Classes in CGAL
Templates in C++

- Function Templates
- Class Templates

Function Templates

```cpp
template <class T>
inline void swap(T& a, T& b) {
    T tmp = a; a = b; b = tmp;
}
```

- T is of the concept Assignable

Class Templates

```cpp
template <class T>
class list {  
    list(const T& x);  
    void push_back(const T& x);  
    typedef ... iterator;  
    iterator begin();  
    iterator end();
};
```

Robustness of Geometry Classes

- Robustness in Geometry
- Templates in C++
- Generic Programming
- Geometry Classes in CGAL

Generic Programming

- Standard Template Library, STL
- Generic Programming
  - Paradigm with focus on algorithm design [Musser89]
- Example: algorithm - iterator - container
- Iterator categories
  - (input | output) < forward < bidirectional < random access

Generic Programming (2)

```cpp
template <class InputIterator, class OutputIterator>
OutputIterator copy(InputIterator first,  
                    InputIterator beyond,  
                    OutputIterator result) {
    while (first != beyond)  
        *result++ = *first++;
    return result;
}
```

```cpp
int a1[100];  
int a2[100];
// ... initialize elements of a1.
copy(a1, a1+100, a2);
```
Generic Programming (2)

```cpp
list<int> ls1;
list<int> ls2;
copy(ls1.begin(), ls1.end(), ls2.begin());
```

Generic Programming (2)

```cpp
list<int> ls1;
list<int> ls2;
copy(ls1.begin(), ls1.end(), back_inserter(ls2));
copy(istream_iterator(cin), istream_iterator(), ostream_iterator(cout, "\n"));
```

Generic Programming Summary

- **Focus on algorithm design**
- **Templates and inline functions in C++**
  - support flexibility at compile time
  - efficiency at runtime
  - flexibility at small scale possible

Robustness of Geometry Classes

- **Robustness in Geometry**
- **Templates in C++**
- **Generic Programming**
- **Geometry Classes in CGAL**

Geometry Classes in CGAL

- **Exact Arithmetic**
- **Coordinate Representation**
- **Degeneracy Handling**

Computational Geometry Algorithms Library

- **Project Goal**
  - make the large body of geometric algorithms developed in the field of computational geometry available for industrial applications.
Sites Developing CGAL

Max-Planck-Institut für Informatik (Germany),
Universität des Saarlandes (Germany),
ETH Zurich (Switzerland),
Freie Universität Berlin (Germany),
INRIA Sophia Antipolis (France),
Martin Luther-Universität Halle-Wittenberg (Germany),
Tel Aviv University (Israel),
Utrecht University (The Netherlands).

Motivation

• Advanced algorithms and complicated data structure
• Complications with robustness
• Complications with degeneracy handling

CG Impact Task Force Report 1996:
Application Challenges to Computational Geometry
Recommendation # 1: “production and distribution of usable (and useful) geometric codes”
[CG-Report96]

Deliverables and Supported Platforms

• C++ source code (templates) (~150 KLOC)
• OS: IRIX 6.5, Solaris 2.6, Linux 2.x,
MS Windows 95/98/NT4
• Compiler: SGI Mips(Pro) CC 7.3, GNU g++ 2.95, Egcs 1.1.2, MS Visual C++ 6.0
• Supports: STL, LEDA, GMP, GeomView,
Inventor, VRML output, ...
• Manuals: LaTeX, for print and HTML

A First Example

Delaunay Triangulation

```cpp
int main () {
    // random points
    CGAL::Random rnd(1);
    CGAL::Random_points_in_disc_2<Point> rnd_pts( 1.0, rnd);
    Delaunay_triangulation dt;
    // triangulation
    CGAL::copy_n( rnd_pts, 100, std::back_inserter( dt));
    leda_window* window =
    // window output
    CGAL::create_and_display_delaunay_triangulation();
    *window << dt;
    delete window;
    return 0;
}
```

Structure of CGAL

<table>
<thead>
<tr>
<th>Basic Library</th>
<th>Algorithms and data structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric Kernel</td>
<td>Geom. primitives, predicates, operations</td>
</tr>
<tr>
<td>Core Library</td>
<td>Configuration, assertions, ...</td>
</tr>
</tbody>
</table>

Support Library

Visualization,
File I/O,
Number types,
Generators,
...
**Geometric Kernel**
- Point, Vector, Direction
- Line, Ray, Segment
- Triangle
- Iso_rectangle
- Bbox
- Circle
- Affine transformation
- Order predicates
- Orientation test
- Incircle test
- Constructions
  - centerpoint
- Intersection
- Squared distance

**Geometric Kernel (2)**
- Exact arithmetic
- Explicit handling of degeneracies

**Exact Arithmetic: Number Types**
- Built-in: double, float, int, long, ...
- CGAL: Filtered_exact, Interval_nt, ...
- LEDA: leda_integer, leda_rational, leda_real
- Gmpz: CGAL::Gmpz
- CLN: cl_I, cl_RA
- others are easy to integrate

**Coordinate Representation**
- Cartesian \( p = (x,y) \)
  - CGAL::Cartesian<Field_type>
- Homogeneous \( p = (x,y,w) = (x/w, y/w) \)
  - CGAL::Homogeneous<Ring_type>
- Geometric primitives parameterized with coordinate representation
  - CGAL::Point_2<CGAL::Cartesian<Field_type>>

**A Start: Support for k-th Roots and static expressions**
```cpp
#include <CGAL/Cartesian.h>
#include <CGAL/Cartesian.h>
#include <CGAL/Arithmetic_filter.h>
#include <CGAL/leda_real.h>
#include <CGAL/Point_2.h>

typedef CGAL::Filtered_exact<double, leda_real> NT;
typedef CGAL::Cartesian<NT> Rep;
typedef CGAL::Point_2<Rep> Point;

int main() {
  Point p(0.1, 0.2);
}
```

**Explicit Handling of Degeneracies**
- Intersection of two segments
- Voronoi diagram of points in the plane
Intersection of two Segments

#include <CGAL/intersections.h>
#include <CGAL/intersections.h>

int main() {
    Segment s(Point(1,1), Point(1,5));
    Segment t(Point(1,3), Point(1,8));
    CGAL::Object result = CGAL::intersection(s, t);
    Point pt;
    Segment seg;
    if (CGAL::assign(pt, result))
        std::cout << "Intersection point = " << pt << std::endl;
    else if (CGAL::assign(seg, result))
        std::cout << "Intersection segment = " << seg << std::endl;
    else
        std::cout << "no intersection" << std::endl;
}

Voronoi Diagram of Points in the Plane

Voronoi Diagram of Points in the Plane

Delaunay_triangulation d;
leda_window* window; // ...
Delaunay_triangulation::Edge_iterator e = d.edges_begin();
while (e != d.edges_end()) {
    CGAL::Object o = d.dual(e);
    Delaunay_triangulation::Segment_iterator seg =
    Delaunay_triangulation::Ray ray;
    if (CGAL::assign(seg, o))
        *window << CGAL::RED << seg;
    else if (CGAL::assign(ray, o))
        *window << CGAL::VIOLET << ray;
    ++e;
}

Implementing CGAL::Object

Implementing CGAL::Object

Summary Part I

- Exact arithmetic
- Templates to select arithmetic
  - tradeoff between efficiency and correctness
  - tradeoff between efficiency and required arithmetic
- Explicit degeneracy handling
  - Polymorphic return type
  - Local design solution, NO global class hierarchy
Introduction to Scene Graphs

- Scene Graph Motivation (Ghali)
- Introduction to Scene Graphs (Blythe)
- Building Scene Graphs (Sowizral)
- Shapes and Transformations in CGAL (Kettner)

Scene Graph Motivation
Sherif Ghali
Max Planck Institute

Organizing Scenes Hierarchically

Creating Scene Hierarchy

- Creating hierarchy procedurally
  - Sequence of commands
  - Commands repeatedly executed
- Creating hierarchy using objects
  - Set up data structures
  - Render by parsing data structures

Creating Hierarchy Procedurally
Creating Hierarchy Procedurally

- Creating hierarchy procedurally
  - Sequence of commands
  - Commands repeatedly executed
- Creating hierarchy using objects
  - Set up data structures
  - Render by parsing data structures

Creating Scene Hierarchy

```java
// Setting up matrices
pushMatrix();
multMatrix(T0);
draw geometry;
popMatrix();
```

Creating Hierarchy Using Objects

```javascript
// Setting up matrices
pushMatrix();
multMatrix(T1);
draw geometry;
popMatrix();
```

```javascript
// Setting up matrices
pushMatrix();
multMatrix(T2);
draw geometry;
popMatrix();
```
Nodes of the Graph

Nodes = \{T_0, T_1, T_2, S_1, S_2\}

Edges of the Graph

Nodes = \{T_0, T_1, T_2, S_1, S_2\}

Edges = \{(T_0,T_1), (T_1, C_1), (T_0,T_2), (T_2, C_2)\}

Edges are (almost!) not needed

A Binary Tree Implementation

class Node {
    Content* c;
    Node* left;
    Node* right;
    ...;
}

- Inefficient access to children

An N-ary Tree Implementation

class Node {
    Content* c;
    vector<Node*> children;
    ...;
}

- Number of children
- Child deletion

Directed Acyclic Graphs

Choice of a Node Class Hierarchy

identical geometry
Object Hierarchy and Class Hierarchy

Possible Class Hierarchy (1)

Possible Class Hierarchy (2)

Children of Transform Groups

Code Example

```cpp
group root = new group;
    translateGroup tgrp1 = new translateGroup(Vector3d(1,0,0));
    translateGroup tgrp2 = new translateGroup(Vector3d(1,0,0));
    shape* cube = new shape("cube.vpl");
    shape* cone = new shape("cone.vpl");
    root->addChild(tgrp1);
    root->addChild(tgrp2);
    tgrp1->addChild(cube);
```
### Code Example

```cpp
Group* root = new Group;
TranslateGroup* tgrp1 = new TranslateGroup(Vector3d(1,0,0));
TranslateGroup* tgrp2 = new TranslateGroup(Vector3d(-1,0,0));
Shape* cube = new Shape("cube.vpl");
Shape* cone = new Shape("cone.vpl");
root->addChild(tgrp1);
root->addChild(tgrp2);
tgrp1->addChild(cube);
tgrp2->addChild(cone);
```

### Traversing a Scene Graph

#### Visitor

- **Transformation node**
  - update transformation matrix
- **Shape node**
  - draw

### Render Visitors

### Introduction To Scene Graphs

**David Blythe**  
**RouteFree**

### Scene Graphs

- What is your objective?
Scene Graphs

- **Graph Representation**
  - What do edges mean?
  - Inherit state along edges
    - group all red object instances together
    - group logical entities together
    - parts of a car
  - Capture intent with the structure

- **Inheritance - Overloaded Term**
  - Behavior inheritance (subclassing)
    - Benefit of OO design
  - Implementation inheritance
    - Perhaps provided by implementation language
    - *Not essential* for a good API design
  - Implied inheritance
    - Designed into the API

- **Inheritance - Example**
  - Ray tracer scene objects
    - sphere, cube, NURBS surface, triangle mesh
  - Each object type supports a `rayIntersect` method
    - Inherit behavior (methods) from base class
    - Implementation of `rayIntersect` method not inherited
  - Object instances arranged in a spatial data structure
    - inherit spatial information from graph

- **Ray Tracer Scene**
  - Object instances inherit spatial information through graph edges

- **Scene Elements**
  - **Interior Nodes**
    - Have children that inherit state
      - transform, lights, fog, color, ...
  - **Leaf nodes**
    - **Terminal**
      - geometry, text
    - **Attributes**
      - Additional sharable state (textures)
Scene Graphs

- Scene Element Class Hierarchy
  - SceneElement
    - Node
    - Geometry
    - Appearance
    - GeometryTrait
    - Group
    - Shape
    - FaceSet
    - Colors
    - Coords

Scene Graphs

- Traversal
  - Perform operations on graph with traversal
    - Like STL iterator
    - Visit all nodes
    - Collect inherited state while traversing edges
    - Also works on a sub-graph

Scene Graphs

- Traversal
  - Typical operations
    - Render
    - Search (pick, find by name)
    - View-frustum cull
    - Tessellate
    - Preprocess (optimize)

Scene Graphs

- Graph Organization
  - Tree structure best
    - No cycles for simple traversal
    - Implied depth-first traversal (not essential)
    - Includes lists, single node, etc as degenerate trees
    - If allow multiple references (instancing)
      - Directed acyclic graph (DAG)
      - Difficult to represent cell/portal structures

Scene Graphs

- State Inheritance
  - General (left to right, top to bottom, all state)
    - Open Inventor
    - Need Separator node to break inheritance
    - Need to visit all children to determine final state
  - Top to bottom only
    - IRIS Performer, Java3D, ...
    - State can be determined by traversing path to node

Scene Graphs

- What State To Inherit?
  - Affects graph organization
    - Affects query performance
    - Red objects together?
    - Parts of a car together?
    - Spatially close objects together?
  - What are the most frequent operations?
Scene Graphs

- Limited State Inheritance
  - Transform hierarchy
  - IRIS Performer, Java3D, ...
  - Allows for efficient spatial queries
    - View-frustum culling
    - Ray-disk
    - Intersection testing
  - Allow secondary inheritance of other attributes
    - Traversal mask?
    - Environment (lights, fog, ...)

- Appearance Overrides
  - One attempt to solve the “highlighting” problem
    - After picking an object, want to display it differently
    - Don’t want to explicitly edit and restore its appearance
    - Use override node in the scene graph to override appearance of children
    - Only works if graph organization matches model organization

- Multiple Referencing (Instancing)
  - Convenient for representing multiple instances of an object
    - rivet in a large assembly
  - Save memory
  - Need life-time management
    - is the object still in use
    - garbage collection, reference counts
Scene Graphs

- Multiple Referencing
  - Changes trees into DAGs
  - Instance of an object represented by its path
  - Difficult to attach instance specific properties
    - e.g., caching transform at leaf node

Scene Graphs

- Paths
  - Node in DAG may have multiple paths to it
  - Record path when remembering specific instances
    - Use path to retrieve inherited properties

Scene Graphs

- Paths
  - Treat path like a mini-scene
  - Run traversals on path (render, search, etc)

Scene Graphs

- Other organizations
  - Logical structure (part, assembly, etc.)
    - Used by modeling applications
  - Topology structure, e.g., boundary
    - Surfaces, faces, edges, vertices
    - Useful for CAD applications
  - Behaviors, e.g., engine graph
  - Environment graph (fog, lights, etc.)

Scene Graphs

- Multiple Graphs (Multi-graphs)
  - Applications may need to organize the data in more than one way
  - May reuse same instances of leaf data
    - Geometry, appearance descriptions
  - New collections of interior nodes
  - Need to keep them all in sync
    - Node additions, deletes

Scene Graphs

- Multi-graphs
  - Use same traversal mechanism to operate on all graph types (including paths)
  - Use a notification/dependency tracking scheme to synchronize graphs
Scene Graphs

- Scene Graphs Are Not Just for Rendering

Building Scene Graphs

Henry A. Sowizral
Sun Microsystems, Inc.
henry.sowizral@sun.com

Overview

- Why scene graphs?
- Scene graph concepts
- Code

Programming At A Higher Level

- Raise the programming “floor”
- Thinking objects ... not vertices
- Thinking content ... not rendering process
- Descriptive

Scene Graph Structure

- Purpose
  - Display
  - Manipulation
  - Animation
  - Interaction
Building A Scene Graph

- A scene graph is a “tree”
  - A hierarchy containing scene data
  - Parents: group children and other parents
  - Children: shapes, lights, sounds, etc.

Scene Graph Example

Scene Graph Items

Scene Graph Groupings

Scene Graph Final View

The Scene Graph
Items in a Scene Graph

- Geometry
- Lights
- Sounds
- Environmental (fog, background)
- Behaviors
- Bounds
- ViewPlatforms — places the user can be

Coordinate systems

Placement

Scene Graph Components

Building Nodes

- Build nodes by instancing Java 3D classes
  Shape3D myShape1 = new Shape3D(myGeom1, myAppear1);
  Shape3D myShape2 = new Shape3D(myGeom2);

- Invoking methods to modify nodes
  myShape2.setAppearance(newAppear2);

Building the Graph

- Building a group of nodes
  BranchGroup myGroup = new BranchGroup();
  myGroup.addChild(myShape1);
  myGroup.addChild(myShape2);
The Constructed Scene Graph

MyGroup

MyShape1
MyShape2

Scene Graphs May Be Large

Hints for Designing Scene Graphs

• Always draw the scene graph
• Structure for the task at hand

What About Encapsulation?

• Subclassing Branch Group
  • Have it construct the requisite additional objects
  • Provide accessor and manipulator methods on the subclassed object

A Scene Graph

Transform 1
Transform 2
Transform 3
Transform 4
Geometry 1
Geometry 2
Geometry 3

Transform classes

• Places the geometry
• Is it part of the geometry?
• Speed
  • Internal classification
Drawing Modes

- Immediate --- no need for scene graph
- Retained --- a scene graph (or display lists)
- Compiled-retained --- optimized scene graphs

Scene Graph Compilation

- Knowledge of use (or hints (or assumptions))
- Tree flattening
- Attribute batching
- Tension
  - Combining geometries
  - Breaking geometries into multiple pieces

Drawing The Scene Graph

- Traversal of scene graphs

Ancillary Structures

Geometric Shapes and Transformations in CGAL

Lutz Kettner
UNC Chapel Hill
kettner@cs.unc.edu

- Overview of the Basic Library
- Separating Basic Library and Kernel
  - geometric traits classes
- Separating Topology and Geometry
  - halfedge data structure
- Affine Transformations
Geometric Shapes and Transformations in CGAL

- Overview of the Basic Library
- Separating Basic Library and Kernel
  - geometric traits classes
- Separating Topology and Geometry
  - halfedge data structure
- Affine Transformations

Basic Library: Triangulation

- Triangle-based data-structure
  - compact, fast, but point location is slow
- Delaunay/Voronoi
- Constrained Delaunay
  - => terrain triangulations
- Regular triangulations
- Tetrahedrization in 3D

Basic Library: Convex Hull

- 5 different algorithms for 2D
- Randomized incremental algorithm for 3D

Basic Library: Geometric Optimization

- Smallest enclosing circle and ellipse in 2D
- Smallest enclosing sphere in dD
- Rectangular p center, 2 <= p <= 4

Basic Library: Search Structures

- Range-, segment-, KD-tree
- Arbitrary dimension
- Mixed segment-range-trees
- Static
- Window query, enclosing query

Basic Library: Planar Map

- Halfedge data-structure
- Efficient point location, fully dynamic
- Arrangements
Basic Library: Polyhedral Surface

- Halfedge data-structure
- Orientable 2-manifolds with boundary

Halfedge data-structure

Geometric Shapes and Transformations in CGAL

- Overview of the Basic Library
- Separating Basic Library and Kernel
  - geometric traits classes
- Separating Topology and Geometry
  - halfedge data structure
- Affine Transformations
Separating Basic Library and Geometric Kernel

Convex_hull
Point_2
leftturn(p,q,r)

CGAL Geometric Kernel
CGAL::Point_2...
CGAL::leftturn(p,q,r)

Combinatorics
Geometry

Custom Traits
Point_2...
leftturn(p,q,r)...

Type Mapping

Function Objects (Functor)

Function Objects (Functor)

template <class Iterator, class Comp>
void sort( Iterator first, Iterator beyond, Comp cmp );

bool compare( string s, string t ) { return strcmp(s,t) < 0; }

vector<string> v;
sort( v.begin(), v.end(), compare );
Function Objects (Functor)

```cpp
template <class Iterator, class Comp>
void sort( Iterator first, Iterator beyond, Comp cmp);
```

```cpp
struct Compare
{
  bool operator()( string s, string t) {
    return strcmp(s, t) < 0;
  }
};
```

```cpp
vector<string> v;
sort( v.begin(), v.end(), Compare());
```

```cpp
struct Compare2
{
  int* counter;
  Compare2( int* p) : counter(p) {}
  bool operator()( string s, string t) {
    ++ *counter;
    return strcmp(s, t) < 0;
  }
};
```

```cpp
int count = 0;
sort( v.begin(), v.end(), Compare2( &count));
```

Function Objects (Functor) (3)

Composing functors

```cpp
int main( int argc, char** argv) {
  if ( argc != 2)
    throw( "usage: remove_if_divides integer 
");
  remove_copy_if( istream_iterator<int>(cin),
                  istream_iterator<int>(),
                  ostream_iterator<int>(cout, "n"),
                  not1( bind2nd( modulus<int>(),
                                atoi( argv[1]) )));
  return 0;
}
```

Currying in C++! All at Compile Time!

An Example: CGAL Convex Hull with LEDA Kernel

```cpp
int main() {
  std::list<Point> pts;  // ...
  std::list<Point> ch;
  CGAL::ch_graham__graham_andrew( pts.begin(), pts.end(),
                                 std::back_inserter(ch),
                                 Traits());
  // ...
}
```

An Example: CGAL Convex Hull with LEDA Kernel (2)

```cpp
typedef leda_rat_point   Pt;
struct My_Less_My_Less_xyxy
{
  bool operator()( const Pt& p, const Pt& q) {
    return (p.xcoord() <  q.xcoord()) ||
           (p.xcoord() == q.xcoord() &&
            p.ycoord() < q.ycoord());
  }
};
```

```cpp
struct My_My_Leftturn
{
  bool operator()( const Pt& p, const Pt& q, const Pt& r) {
    return left_turn( p, q, r); // leda_left_turn
  }
};
```

An Example: CGAL Convex Hull with LEDA Kernel (3)

```cpp
struct Traits
{
  typedef Pt Point_2;
  typedef My_Less_My_Less_xyxy Less_xy;
  typedef My_My_Leftturn Leftturn;

  Less_xy get_less_xy_object() const { return Less_xy(); }
  Leftturn get_leftturn_object() const { return Leftturn(); }
};
```

Member functions required for functors with state
Geometric Shapes and Transformations in CGAL

- Overview of the Basic Library
- Separating Basic Library and Kernel
  - geometric traits classes
- Separating Topology and Geometry
  - halfedge data structure
- Affine Transformations

Separating Topology from Geometry

- Topology
  - Topological_map
- Geometry
  - Planar Map
  - Halfedge_data_structure
- Polyhedron
- Vertex
- Halfedge
- Face

Separating Topology from Geometry (2)

Polyhedron
- provides ease-of-use
- protects combinatorial integrity
- redefines vertex, halfedge, face

Halfedge_data_structure
- manages storage (container)
- defines handles and iterators

Items
- stores actual information
- Vertex
- Halfedge
- Face

Cyclic Type Dependencies with Templates

Graph
- Node
- Edge

Cyclic Type Dependencies with Templates

Graph
- Node<\text{G}>
- Edge<\text{G}>

Graph<\text{N,E}>

Node<\text{G}>

Edge<\text{G}>
Cyclic Type Dependencies with Templates (2)

```cpp
template <class Graph>
struct Node
{
    typedef typename Graph::Edge Edge;
    Edge* edge;
    // .... maybe some more edges ....
};
```

Cyclic Type Dependencies with Templates (3)

```cpp
template <class Graph>
struct Edge
{
    typedef typename Graph::Node Node;
    Node* node;
};
```

Cyclic Type Dependencies with Templates (4)

```cpp
int main() {
    typedef Graph
    struct Node : public Graph
    {
        typedef typename Node
        Edge* edge;
        // .... maybe some more edges ....
    };

    Node node;
    Edge edge;
    node.edge = &edge;
    edge.node = &node;
}
```

Cyclic Type Dependencies with Templates (5)

```cpp
template <class Graph>
struct Colored_node : public Node
{
    int color;
};
```

Example: Declaration of Default Polyhedron

```cpp
#include <CGAL/Cartesian.h>
#include <CGAL/Polyhedron_default_traits_3.h>
#include <CGAL/Polyhedron_3.h>

typedef CGAL::Cartesian<double> Rep;
typedef CGAL::Polyhedron_default_traits_3<Rep> Traits;
typedef CGAL::Polyhedron_3<Traits> Polyhedron;

int main() {
    Polyhedron P;
    Polyhedron::Halfedge_handle h = P.make_tetrahedron();
    assert(P.is_tetrahedron(h));
    return 0;
}
```

Geometric Shapes and Transformations in CGAL

- Overview of the Basic Library
- Separating Basic Library and Kernel
  - geometric traits classes
- Separating Topology and Geometry
  - halfedge data structure
- Affine Transformations
Affine Transformations

- Transforms points, vectors, and hyperplanes (covectors)
- Functor (STL)
  ```cpp
  std::transform(P.points_begin(), P.points_end(), P.points_begin(), AA);
  ```
- Smart pointers with reference counting
- Polymorph
  - specialized representations for:
    - translation, rotation, uniform scale

Summary Part II

- Templates, C++, and concepts from generic programming (iterators, functors)
  - provide expressiveness
  - provide flexibility at compile time
  - provide efficiency at runtime
- Shapes beyond IndexedFaceSet
  - e.g., for subdivision surfaces, mesh compr., etc.
- Example code on the CD-Rom

Acknowledgements

Thanks to all my friends and colleagues in CGAL!
Thanks to my advisors Emo Welzl and Jean-Daniel Boissonnat.

Support from the ESPRIT IV LTR Projects No. 21957 (CGAL) and 28155 (GALIA), and by the Swiss Federal Office for Education and Science (CGAL and GALIA) is acknowledged.

Internet Addresses

- [SGI-STL]
  SGI: Standard Template Library Programmer’s Guide:
  http://www.sgi.com/Technology/STL/

- [CGAL]
  Computational Geometry Algorithms Library:
  http://www.cgal.org

- [LEDA]
  Library of Efficient Data Types and Algorithms:
  http://www.mpi-sb.mpg.de/LEDA/

References

- [Austern98]

- [CG-Report96]

- [Edelsbrunner90]
References

[Emiris97]

[Fabri99]

[ISO98]

References

[Kettner98]

[Kettner99]

[Musser95]

References

[Schirra99]

[Stepanov95]

[Stroustrup97]

[Weihe98]

Lighting and Visibility

- Camera and Light Objects (Ghali)
- Performing Visibility Computation (Blythe)
- Separating the Physical and the Virtual Worlds (Sowizral)

Camera and Light Objects

Sherif Ghali
Max Planck Institute
The Camera

• Non scene-graph issues
  • Specifying the camera
• Scene-graph issues
  • Relation of camera object to the scene graph?
  • Transformations affecting the camera?

Non-Scene Graph Issue: Specifying the Camera

Camera Object: Single Class

Camera Object: Orthographic and Perspective Classes

The Camera

• Non scene-graph issues
  • Specifying the camera
• Scene-graph issues
  • Relation of camera object to the scene graph?
  • Transformations affecting the camera?
The Camera

- Non scene-graph issues
  - Specifying the camera
- Scene-graph issues
  - Relation of camera object to the scene graph
  - Transformations affecting the camera

Is the Camera a Node in the Scene Graph?

1. User is immersed in the virtual world
   - Camera is a node in the scene graph
2. User is looking (through a window) at the virtual world
   - Camera is not a node in the scene graph

Option 1: Camera Is Part of the Scene Graph

Option 2: Camera Is Not Part of the Scene Graph

The Camera

- Non scene-graph issues
  - Specifying the camera
- Scene-graph issues
  - Relation of camera object to the scene graph
  - Transformations affecting the camera

Transformations Affecting the Camera
Transformations Affecting the Camera

Light Sources

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light

Non Scene-Graph Issue: Specifying the Light

- Light type:
  - Directional light:
    - Light at infinity
  - Point light:
    - Light at proximity
  - Spot light:
    - Specific direction and spread angle
- Light intensity
- Light color

Light Classes

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light

Light Sources

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light
**Light Sources**

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light

---

**Subset of Scene Graph Illuminated**

1. Light illuminates nodes traversed after it:
   1. Scene Graph parsed sequentially
   2. Scene Graph parsed hierarchically
2. Light illuminates a region of influence

---

**Subset of Scene Graph Illuminated**

1. Light illuminates nodes traversed after it:
   1. Scene Graph parsed sequentially
   2. Scene Graph parsed hierarchically
2. Light illuminates a region of influence

---

**Light Illuminates Geometry Traversed After It**

---

**Light Illuminates Descendant Geometry**

---

**Light Sources**

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light
Subset of Scene Graph Illuminated

1. Light illuminates nodes traversed after it:
   1. Scene Graph parsed sequentially
   2. Scene Graph parsed hierarchically
2. Light illuminates a region of influence

Light Illuminates a Region of Influence

Light Sources

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light

Transformations Affecting Light

Performing Visibility Computation

- Coordinate Spaces
  - Object Space
  - Raw object coordinates
  - World Space
  - After applying modeling transforms
  - Eye Space
  - After applying viewing transform
  - Window Space
  - After projection and viewport transform

Performing Visibility Computation

David Blythe
RouteFree
Performing Visibility Computation

**Coordinate Spaces**
- Some low-level APIs combine modeling and viewing transforms into one
  - OpenGL
- Scene graph APIs separate modeling transforms and viewing transform

---

Caching Visibility Computation

**The Camera Abstraction**
- Typically includes both viewing and projection transforms
  - Position and orientation
  - Projection type (Perspective, Parallel)
  - Depth of Field
  - Mapping to the display surface (viewport)

---

Performing Visibility Computation

**Camera in the Scene Graph**
- Allows tethering the camera to scene objects
- Need to locate camera before rendering
- Multi-surface rendering complicates things
  - Multiple cameras in the scene graph?
Performing Visibility Computation

- **Specifying Parameters**
  - Keep projection and viewing transforms separate
  - Use camera for rendering and view-frustum culling
  - Projection
    - Use frustum as canonical representation
      - frustum (left, right, bottom, top, near, far, perspective)
      - Easy to convert to plane equations for culling

Performing Visibility Computation

- **Specifying Frustum**
  - Additional projection convenience functions
    - Perspective(fovy, aspect, near, far)
    - Ortho(left, right, bottom, top, near, far)
    - Frustum(left, right, bottom, top, near, far)

Performing Visibility Computation

- **Specifying Viewing Transform**
  - Use 4x4 matrix as canonical representation
  - Convenience functions for generating matrix
    - LookAt(eye_point, reference_point, up_vector)

Performing Visibility Computation

- **Specifying Viewport Mapping**
  - Viewport(x, y, width, height)
  - May need to set scissor as well
    - If rendering to a portion of a window
  - Dependency between display surface, viewport, and projection
  - Want to observe aspect ratio
  - Useful to tie all camera parameters together

Performing Visibility Computation

- **Multiple Cameras**
  - Multiple channels on the same accelerator
  - Multiple accelerators
  - Tiled displays
    - Curved or domed
    - Flat
  - Single eye point, multiple cameras
    - Link with offsets

Performing Visibility Computation

- **View-Frustum Culling**
  - Quickly eliminate irrelevant objects (not-visible)
    - Spend less time drawing
    - Test individual objects
      - Use bounding volumes as proxy geometry
    - Use spatial organization and hierarchical bounding volumes
      - Eliminate large sections of data base quickly
Performing Visibility Computation

- **View-Frustum Culling**
  - What space are bounding volumes stored in?
  - Instancing use hierarchical model spaces
  - Incorporate viewing transform during cull operation

Hierarchical Bounding volumes

Performing Visibility Computation

- **Multiple Cameras**
  - Non-tiled displays
  - Top, front, side views (CAD)
  - Orthographic and perspective views
  - Wire-frame and Solid views
  - Non-simple relationship between cameras

Performing Visibility Computation

- **Spatial Queries**
  - Other uses for hierarchical bounding volumes
    - Intersection operations
      - Simple collision detection
      - Terrain following
    - Factoring to two pass operations
      - Rather than replicating culling code in other tasks
      - Cull first, save result, operate on result

Performing Visibility Computation

- **Generalize Frustum Culling**
  - Use polytope rather than frustum
  - Collection of half-spaces (plane equations)
    - e.g., Cylinder -> faceted cylinder
  - More computationally expensive, but tighter culls
Performing Visibility Computation

• Occlusion Culling

Visible Geometry

Occluded Geometry

Model courtesy SDRC

Performing Visibility Computation

• Occlusion Culling
  • Don’t render occluded objects
  • Find list of occluders
    • Good occluder has a large window space extent
    • Transform bounding volume to window space
    • Remember occluders from frame to frame
    • Where to save occluder list?
      – Instancing need paths to occluder geometry

Performing Visibility Computation

• Occlusion Culling
  • Test other objects (PVOs) against occluders
    • Compare depth values and window coverage
      – Is object contained within occluder
    • Use real projection of occluders (not BVs)
    • Use projected bounding volumes of PVOs
    • Use software scan converter or hardware support
      – e.g., Occlusion maps

Performing Visibility Computation

• Visibility Pipeline

View-Frustum Cull

Occlusion Cull

Render

Update camera parameters

Camera-based Model

• Camera-in-the-scene
  • The program controls the camera’s
    • Position
    • Orientation
    • Aspect ratio
    • Field-of-view (FOV)

• The camera’s image maps onto a canvas

Separating The Physical and Virtual Worlds

Henry A. Sowizral
Sun Microsystems, Inc.
henry.sowizral@sun.com

Copyright © 2000 Henry A. Sowizral
**Works Well For Monitors**

- Head-Mounted Displays
- Head-tracked Monitor

**Too much control**

- **FOV**
  - Fixed: physical constraints
  - Changing: function of head position
- **Aspect ratio**
  - Determined by physical screens

**Head-mounted FOV**

**Moving Left, Moving Right**

**Image Plate**
Moving Closer, Moving Further

Too Little Control

- Multiple displays
  - Coplanar (tessellated wall)
  - Non-coplanar (portal: cave-like environment)
- Head position

One Screen Just Isn’t Enough

Surrounding Yourself With Data

- Not a camera-based model
  - ViewPlatform metaphor accommodates head-tracking
- Easily change display environments
  - Scene graph independent
  - Without rewriting code

A New View Model

A User And A Monitor
A Scene Graph

ViewBranch

Separating the Physical from the Virtual

- Virtual World
  - ViewPlatform
  - Position and orientation
  - Navigable

- Physical World
  - Location of screens
  - Location of tracker

In The Virtual World: The ViewPlatform, A Magic Carpet

In The Physical World: The Screens, User, And Tracker

A Portal's ViewBranch
Coexistence

- Physical world
  - User's location and orientation
  - Screen location and orientation
  - Speakers, etc.
- Virtual world
  - Abstract location and orientation
- One-to-one and onto mapping

Relating The Two: Coexistence

Manipulating Your Data

Input and User Interaction

- Scene Graph Inputs (Ghali)
- Input File Formats (Blythe)
- User Interaction (Sowizral)

Scene Graph Inputs

Sherif Ghali
Max Planck Institute
Off-line Storage

- Adopt a file format?
- Adopt a minimalistic file format?
- Save compiled graph?
- Save human-readable file?

Information Saved

- Geometry
  - Vertex and polygon lists
- Geometry and Materials
  - Subset of VRML / OpenInventor
- Geometry, Materials, and Hierarchy
  - VRML / OpenInventor

A Minimalistic File Format

- Advantages
  - Automatic generation is easy
  - Human readable
  - Compact
- Disadvantages
  - No material information
Information Saved

- Geometry
  - Vertex and polygon lists
- Geometry and Materials
  - Subset of VRML / OpenInventor
- Geometry, Materials, and Hierarchy
  - VRML / OpenInventor

Geometry and Materials

Material {  
  diffuseColor  [ 1 0 0, 0 1 0, 0 0 1, 0.7 0.7 0.7 ]  
}  
MaterialBinding {  
  value PER_FACE  
}  

Geometry, Materials, and Hierarchy

Material {  
  diffuseColor  1 0 0  
}  
Separator {  
  Translation {  
    translation  0 0 0  
  }  
  Cylinder ( )  
}  
Separator {  
  Translation {  
    translation  0 1 0  
  }  
  Cone ( )  
}  

**Event Handling**

- User ignores events
- User handles events

**Event-Handling Transformation Node**

- Node
- Transformation
- ActiveTransformation

**Visitors**

- Visitors traverse scene graph
- Render Visitors
  - activate draw methods
- Event Visitors
  - activate handleEvent methods
  - propose events to nodes

**Visitor Classes**

- Visitor
- RenderVisitor
- EventVisitor

**Nodes x Visitors**

- Node
- Transformation
- ActiveTransformation
- render()
- handleEvent()
- update matrix
- ignore
- update matrix

**Visitor-Specific Methods in Nodes**

- Node
- Transformation
- ActiveTransformation
**Adding Nodes / Visitors**

<table>
<thead>
<tr>
<th>update matrix</th>
<th>ignore</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>update matrix</td>
<td>handle event</td>
<td>...</td>
</tr>
</tbody>
</table>

**Code Example**

```c
main(int argc, char ** argv) {
    DirectionalLight *lum = new DirectionalLight(Point3d(3,4,5));
    ActiveTransformGroup *active = new ActiveTransformGroup;
    Shape *cube = new Shape("cube.vpl");
    lum->addChild(active);
    active->addChild(cube);
    Camera cam(Point3d(0,3,3), Point3d(0,0,0), Vector3d(0,1,0));
    window = new Window(argc, argv, lum, cam);
    window->mainLoop();
    return 0;
}
```

**Input File Formats**

- **Data Driven Applications**
  - Viewing applications are data independent
  - Use input files to drive viewer
    - static scenes provide simple scene data
    - dynamic scenes need to encapsulate behavior
      - Inventor/VRML engines, behaviors, ...
  - The database is the application

- **Uses**
  - Restore/Restart
  - Application Interchange
  - Import/Export
  - Prototyping/Debugging
  - Run-time "Paging"

- **Restore/Restart**
  - Save current state of application
    - e.g., authoring application
  - Serialization of objects
    - Serialization built into objects?
  - Typically involves more than scene data
Input File Formats

- **Application Interchange**
  - Share data between different applications
  - Complicated if applications use different APIs
    - Pick an interchange format (VRML?, STEP)
  - If applications use the same API
    - Native file format
      - e.g., Inventor File format

- **Import/Export**
  - Viewers/authoring tools read multiple input formats
    - VRML, OBJ, 3DStudio, Maya, ...
  - Authoring programs export multiple formats for other applications
  - Lossy process
    - Toolkit objects don’t capture all information in file
    - File format can’t capture object semantics

- **Prototyping/Debugging**
  - Very readable, easy to enter
  - ASCII preferred
  - Ideally represent all attributes of Toolkit objects
    - Data types
    - Behavior
    - Native format best match

- **Reader Class Hierarchy**
  class Reader {
    virtual Node* loadFile(const char* filename) = 0;
  };

  class VrmlReader : public Reader {
    Node* loadFile(const char* filename);
  };

  class ObjReader : public Reader {
    Node* loadFile(const char* filename);
  }

- **Example - VRML Format**
  Transform {
    translate 0 0 3
    childen [ 
      Shape {
        geometry Sphere
        appearance Appearance {
          material Material { diffuseColor 0 0 1 } } ]
    ]
  }

- **Use offline translator?**
  - E.g., Performer has pfconv
  - If translator uses the same API code base, should integrate as a reader module
  - Simple reader class hierarchy
    - Find correct reader using filename extension - or -
    - Magic numbers
Input File Formats

- Paging
  - Database too large to fit in memory
  - Segment a database
    - spatial organization
    - tiling?
  - High performance read and write
    - binary representation
      - (pointers, endianness)

Input File Formats

- Paging
  - Organize data for I/O
    - separate arrays from objects
  - read directly into objects
    - avoid copying memory
  - Format may change when object definitions change
    - IRIS Performer
    - Not good as an interchange format

Input File Formats

- Paging
  - Need a way to trigger paging operations from scene
  - ‘External Reference’ node refers to file storage
  - Trigger file read when traverse external reference
  - May want to page out parts of scene when not in view
    - Throw away if scene is immutable
    - Save changes if not

Input File Formats

- Instancing
  - Multiple references to an object definition
  - Preserve using a DEF/USE table (hash table)
  - Allow use before definition?
    - Need to patch on definition

Input File Formats

- Machine Dependencies
  - What if a certain feature isn’t supported?
    - Multi-texture, cube maps?
    - Convert to a different abstraction
  - Author on one platform/Use on another
  - Use more generic terms
    - NICEST, FASTEST
  - Map definitions to specific platform

Input File Formats

- Machine Dependencies
  - Create more generic abstractions
    - Bump map rather than multi-texture config
    - Environment map rather than cube map
    - Using hint to bias mappings
**Input File Formats**

- **Extensibility**
  - New object types added to system
  - Add new definitions to file format
  - Dynamically loaded object types
    - May need file format hook to load file reader extensions

- **Extension Example**

```transform
translate 0 0 3
children [
  Shape {
    geometry Foo {
    
    appearance Appearance {
      material Material { diffusionColor 0 0 1 }
    }
  }
]
```

- **New data type**

- **Reader encounters unknown type Foo**
  - Reader searches registry for module to handle Foo
  - Reader loads module and jumps to Foo handler
  - Maybe add tag to file format to preload extensions

- **How much processing to do in reader?**
  - Generate normals
  - Tessellate geometry
  - Generate mipmap textures
  - If reader does work, may need to pass hints to reader to control processing
  - Harder to do code reuse between readers

- **Post-Reader Processing**
  - Have reader mark scene for additional processing
  - Run “fix-up” traversals after reader completes
  - Move work out of readers into a central place
    - Allow more control
    - Integrate with other optimization processing
      - e.g., re-strip geometry, generate LODs, etc.
    - Need a way for reader to tag the scene

- **Tagging with attributes**

  ```bash
generate mipmap
  + generate normals
  generate mipmap
  + generate normals
```

  ```bash
generate normals
  crease_angle = 45
  ```
Input File Formats

• **Built-in Serialization**
  - File I/O defined as part of object definition
  - Only works for native format
  - Need another way to export to other formats
  - Problematic with paging
    - all object data may not be grouped together

Input File Formats

• **Adding Behaviors to File Format**
  - Holy grail of file formats
  - database is the application
  - Analogue in the database world is “stored procedures”
  - Store processing with the data

Input File Formats

• **Adding Behaviors to File Format**
  - Add serializable form of behavior data types
  - Add behavior object connectivity
    - e.g., Open Inventor engines + routes
    - Routes affectively form another graph (engine graph)
  - May need to dynamically load code to implement behavior
    - Similar to problem of loading code for new data types

User Interaction

Henry A. Sowizral
Sun Microsystems, Inc.
henry.sowizral@sun.com

Synchronous Versus Asynchronous

• Synchronous --- the input must occur now before anything else can happen
• Asynchronous --- the user may request an action at any time
Events As Drivers

• Keyboard presses
• Position sensor motion
• Timers expiring
• Collision occurrence

Model, View, Controller

• Model ~ Data
• View ~ An on-screen representation
• Controller ~ An method for changing aspects of the model through the view

Coupling

• Tight coupling between Controller - View
• Controller has a representation within the View
• Manipulation closely tied to the controller

Input Device Interface

• An abstraction of an input device
  • A 6DOF value
  • Any number of multi-position buttons

Query Mechanism

• Scene graph query mechanism
• Independent of device

Picking

• The pick process
  • PickShape defines the query
  • A “controller” behavior uses the query to search for an answer
• How do you define the PickShape?
  • The mouse
  • Gloved hands
Collision detection

- Generate an event when two geometries collide
  - Behavior driven
  - Sensor driven

Output

- There is more to output than graphics
- Sound!

The Future of Scene Graph APIs

- Scene Graph Shadow Generation (Ghali)
- Multiprocessing and Extensibility (Blythe)
- The Future of Scene Graph APIs (Sowizral)

Scene Graph Shadow Generation

Sherif Ghali
Max Planck Institute

Active vs. Passive Light Sources

- Passive light source:
  - Shadow-responsibility lies on client
- Active light source:
  - Client is relieved from shadow generation

Responsibility for Shadows
Shadow Generation
- Raster light sources
- Object-space light sources
- Fake shadows

Raster Light Sources
- First pass: raster shadow map computation
- Moving objects, camera, light source
- Handling curved objects
- Shadow aliasing

Object-Space Light Sources - Geometry Modification
- Object-space visibility
- One pass to update geometry
- Moving objects or light difficult to handle
- True shadow boundaries

Fake Shadows
- No extra visibility pass
- Geometry is duplicated
- Usually single ground plane

Clipping Active-Light Classes

Clipping Light Sources
Compiling New Geometry

Fake Shadows Active-Light Classes

Fake Shadows Light Sources

Fake Shadows Light Sources

Multi-processing and Extensibility

David Blythe
RouteFree

Multi-processing/Multi-threading

- What is your objective?
Multi-processing/Multi-threading

Motivation
- Improve performance
  - Increase scene complexity
  - Decrease frame time
- Utilize other processors in multi-processor configurations
  - 2-way systems becoming common

Processing Tasks
- Event processing
- Behavior processing
- LOD resolution
- Collision detection
- View-frustum culling
- Sort to minimize state changes
- Render audio
- Render to display surfaces
- Occlusion culling

Approaches
- Parallel Traversal
  - Process parts of scene in parallel
    - ViaKit
  - Pipelined Traversal
    - Perform different tasks on whole scene using thread per task
      - Performer, Fahrenheit/XSG, Java3D
    - Leave up to application :(  
- Perform multiple tasks at each node during traversal
- Break scene up into chunks and assign to threads
- Load balancing complicated
  - Early culling makes it difficult to predict load
  - May involve locking if not careful
    - Locking == bottlenecks

Parallel Traversal
- Can be good for computationally intensive tasks
  - Surface tessellation
  - Mipmap generation
- Very bad for drawing
  - Only one thread can talk to accelerator at a time

Parallel Traversal
- Thread A
- Thread B

Leave up to application :(
Multi-processing/Multi-threading

- Pipelined Traversal
  - Use a thread per task
  - Problem if threads write to nodes
    - Locking == bottlenecks
    - Use multiple logical copies of data
    - Phased pipeline to allow overlapped processing
    - Application computation
    - Cull/LOD Resolution
    - Render

- Multi-buffering
  - Provide illusion of a copy of node data (buffer) for each thread
  - Use copy-on-write for efficiency
    - Node starts off with a single copy
    - Create a copy per thread on first write
    - Don’t consolidate copies
  - Use special processing to synchronize state of buffers

- Update Processing
  - Synchronize an upstream buffer with a downstream one
  - Log all changes to a buffer
    - Don’t need to copy data, just a reference to data
    - Play log against downstream buffer
    - Update downstream buffer using references to upstream buffer

- Serendipity
  - Log can be used for other things
  - Integrate with event processing to implement
    - Sensors/Auditors
  - Transmit log over network to synchronize distributed databases
    - Need actual data, not just reference
**Multi-processing/Multi-threading**

- **Shared Address Space or Not**
  - Older APIs assume fork model
  - Use shared arenas to hold scene data
  - Process-private state easy
  - New APIs use multi-threading with shared address space
    - Pthreads, Java threads
    - Entire address space shared
    - Thread-private state challenging

- **Hybrids**
  - Pipeline + Parallel Traversal
    - Within one pipeline stage perform parallel traversal
    - Good for computationally intensive tasks
      - Load balancing simpler

**Extensibility & Component Replacement**

- **What is your objective?**
  - Performance
  - Generality
  - Speed of light

- **Motivation**
  - Designers can’t foresee all necessary functionality
  - Add new types of traversal tasks
    - Collision detection
    - Occlusion culling
  - Add new scene abstractions
    - Advanced shading
    - Surfaces
    - Progressive meshes
Extensibility & Component Replacement

**Motivation**
- May want to customize a component
  - Add additional data to a node
  - Perform additional processing during a traversal

Cheesy™ Extensibility
- Add "user-data" pointers to nodes
  - Store additional application-specific data at node
- Add traversal callbacks to nodes
  - Perform application-specific processing at each node visit
- Hard to share with other developers
- Extensions embedded in the application

Open Source It!
- Customize to your hearts content
- Can change anything
- Very difficult to share
  - Multiple incompatible customizations
- Very difficult to maintain correctness
  - May make random changes to semantics

Fahrenheit Approach
- Kernel facilities with extensibility model
  - Multi-buffering, error handling, node, traversal
- All built-ins above kernel use extension services
  - Everything is an extension
- Uses COM-hybrid for binary compatibility
  - Break some rules to improve performance
  - Extenders deal with

Binary Compatibility
- Extensibility + Compatibility challenging
- Extra hard with C++
  - private mixed with public
  - use discipline - or -
  - solvable using methodology like COM
  - Easier in Java
- Still need to watch out for semantic changes

Fahrenheit Approach
- Everything a component
  - Interface and implementation kept separate
  - Packaged in dynamic libraries
  - Advertised in a database
- Components loaded on demand
- Components must obey rules
### Extensibility & Component Replacement

- **Fahrenheit Approach**
  - Add new nodes, traversals, other components
  - Replace implementations of components with alternatives
    - e.g., different culling algorithms
    - different tessellation algorithms
  - Create different pipelines
  - Easy to share components with other developers

- **Hard Problems**
  - Fragile base classes
    - Resist temptation to customize base classes
    - Re-implement all subclasses
  - Left with user-data, RFields
  - Performance cost for encapsulation
    - Virtual function calls
    - No inlines

---

### Scene Graph API Futures

Henry A. Sowizral
Sun Microsystems, Inc.
henry.sowizral@sun.com

---

### Graphics Outpacing CPU

- Log Performance vs. Time
- Graphics vs. CPU
The Hardware Gaps

- CPU versus graphics
- Bus versus graphics
- Memory versus graphics

Historic Trends

- Back to the Past
- Display lists
  - Vector displays
  - Raster displays
  - Equilibrium

The Software Gap

- Hardware evolving faster than software
- Time-to-market dominates

3D Software Development

- Newer graphics hardware brings features and complexity
  - Most developers ignore HW-specific features
  - Over 95% of SGI Performer apps are built with Perfly, the included demo app
  - Developers struggling with:
    - Texture mapping, anti-aliasing, multi-pass textures, fog, bump mapping, . . .
    - They will face parallelism, texture streaming, asynchronous culling, morphing, multi-channel, multi-pipeline configurations, . . .
  - Rising development complexity
    - ~8 years ago: Castle Wolfenstein, 6 months to develop, $100K, no tools
    - ~4 years ago: Doom, 12 months, $300-$500K, OpenGL
    - ~2 years ago: Quake, 18 months, > $1M, OpenGL, Glide
    - In ~2 years: 24-36 months development time, > $10M, scene graph APIs
  - Hardware development cycles are accelerating!

All Future Graphics Programs Will Use Scene Graphs

- This is true today: but mainly custom
- Time-to-market dominates
- Scene graphs:
  - specialized versus generic

Problems With Generalized Scene Graph

- Memory usage
- Speed of hardware evolution
Missing Features

- Support for modeling
- Complex illuminations
- Complex material interactions

Issue: Composition of Worlds

- Locality of structure and reference
- Encapsulation

Stability and Lifecycle of a Graphics API

- Functionality
- Extensibility

Functionality

- Keeping pace with hardware
- Hardware trends
  - Speed: 100Mtris/sec
  - Surfacing
  - Multi-textures nee programmable shaders

Extensibility

- Traversal with pre-, in-, and post-order visit
  - Integration and use of low level API
  - Low-level API specific
  - Registration

What To Do?

- Plan for a variety of
  - Display environments
  - Interaction modalities
- Plan for multiple
  - Processors
  - Platforms
- Plan for change