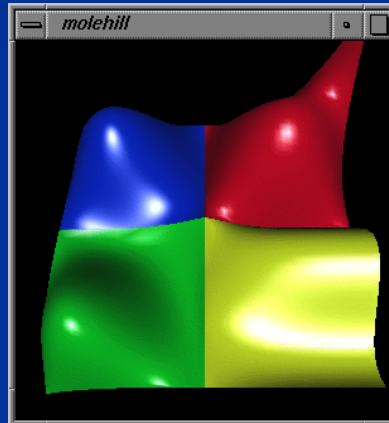


Curves and Surfaces in OpenGL



Felipe Coelho

Introduction

- OpenGL supports Bezier curves and surfaces
 - Evaluators
- Generate 1, 2, 3, or 4-D curves and surfaces
- Curves use 1-D evaluators
- Surfaces use 2-D evaluators
- GLU library
 - Uses this mechanism in 4-D to provide NURBS curves.

Bezier Curves

- 1-D evaluator function:

```
glMap1f(type, u_min, u_max, stride, order,  
        point_array)
```

- Each one of these types must be enabled by `glEnable(type)`

- Evaluate with `glEvalCoord1f(u)`

Example

- Consider a cubic Bezier curve over $(0, 1)$

```
point data[ ]={.....}; * /3d data /*  
glMap1f(GL_MAP_VERTEX_3,0.0,1.0,3,4,data);  
  
glEnable(GL_MAP_VERTEX_3);
```

Equally Spaced Points

- Suppose we have defined evaluators for the cubic Bezier curve over $(0,10)$ and the array of control points

```
glBegin(GL_LINE_STRIP)
    for (i = 0; i < 100; i++)
        glEvalCoord1f ((float) i / 100.0)
glEnd();
```

- Using OpenGL

```
glMapGrid1f (100, 0.0, 10.0);
glEvalMesh1(GL_LINE, 0, 100);
```

Bezier Surfaces

- 2-D evaluator function:

```
glMap2f (type, u_min, u_max, u_stride, u_order,
         v_min, v_max, v_stride, v_order,
         pointer_to_data)
```

- Evaluate with `glEvalCoord2f(u,v)`

Example

- Consider bicubic Bezier surface over $(0, 1) \times (0, 1)$

```
point data[4][4]={.....};
glMap2f(GL_MAP_VERTEX_3, 0.0, 1.0, 3, 4, 0.0,
        1.0, 12, 4, data);

glEnable(GL_MAP_VERTEX_3);
```

Rendering Lines

```
for(j = 0; j < 100; j++)
{
    glBegin(GL_LINE_STRIP);
        for(i = 0; i < 100; i++)
            glEvalCoord2f((float) i/100.0, (float)
                          j/100.0);
    glEnd();
    glBegin(GL_LINE_STRIP);
        for( i = 0; i < 100; i++)
            glEvalCoord2f((float) j/100.0, (float)
                          i/100.0);
    glEnd();
}
```

NURBS Functions

- Can use evaluators in 4-D using the GLU library in OpenGL
 - `gluNewNurbsRenderer`
 - `gluNurbsProperty`
 - `gluBeginNurbsSurface`
 - `gluNurbsSurface`
 - `gluEndNurbsSurface`
- **Trimming curves**

Quadrics

- OpenGL supports several quadric objects
 - disks, cylinders, spheres
- Objects can be transformed
- Quadrics routines use polygonal approximations where the application specifies the resolution

Sources

Interactive Computer Graphics: A Top-Down Approach Using OpenGL, Fourth Edition. Edward Angel. Pearson Education, 2005.

OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 2 (5th Edition). OpenGL Architecture Review Board, Dave Shreiner, Jackie Neider, Mason Woo, Tom Davis. Addison-Wesley.

Questions?

Application of Computer Graphics in Manufacturing Diagnosis

Na Ni
2006.12.14

Outline

- Introduction
- Problem definition
- Fault Localization using ICP algorithm
- Demo

Introduction

- **Fault diagnosis** of industrial components becomes increasingly important for improving the quality of manufacturing and reducing the cost for product testing.

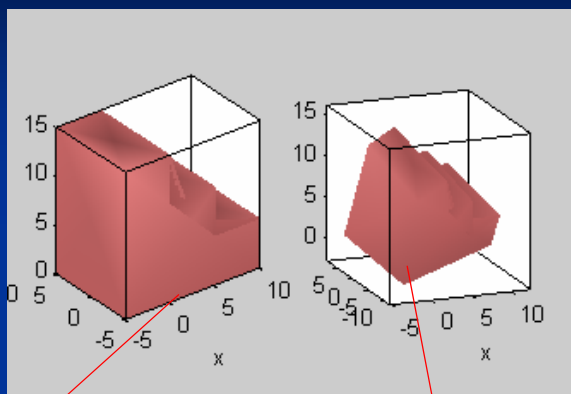
Fault Diagnosis

- What is the fault?
 - Shape
 - Orientation
 - Position
- Where do they come from?
stamping error, tooling error

Task

- Diagnostics Task
 - ❖ Fault Identification
 - ❖ **Fault Localization**
 - ❖ Fault Isolation

Fault Localization



Ideal model

True model

Error :

Shape -> Noise on
sampling points

Orientation ->
Rotation

Position ->
Translation

My Task

- Find the Rotation, Translation Matrix
- Align the true model to the ideal model
- Point out the error

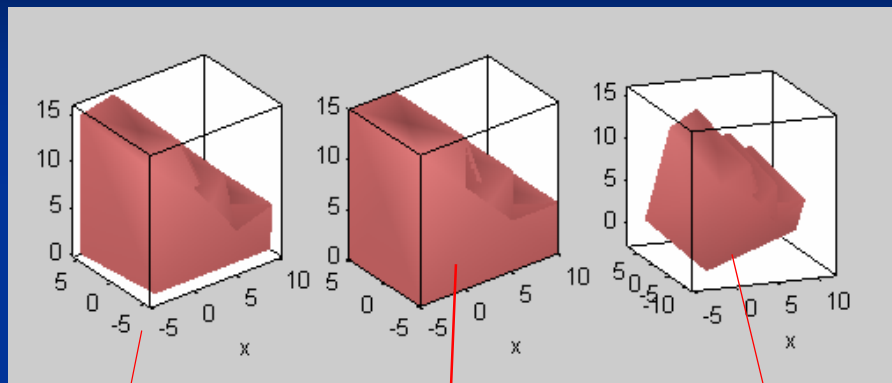
ICP Algorithm

- Iterative Closest Point (ICP) is an algorithm employed to match two clouds of points.
- It iteratively estimates the transformation (translation, rotation) between two raw scans.
- Inputs: two raw scans, initial estimation of the transformation, criteria for stopping the iteration.
- Output: refined transformation.

ICP Algorithm

- Essentially the algorithm steps are:
 - Associate points by the nearest neighbor criteria.
 - Estimate the parameters using a mean square cost function.
 - Transform the points using the estimated parameters.
 - Iterate (re-associate the points and so on).

Demo

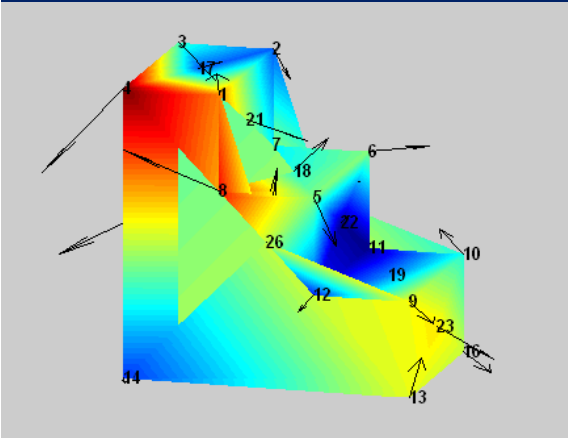


Result after using ICP

Ideal model

True model

Demo



1. Every 1×3 vector is always assumed to be an RGB triplet specifying a color directly.

2. The length of the arrow is assumed to be the error of that point.

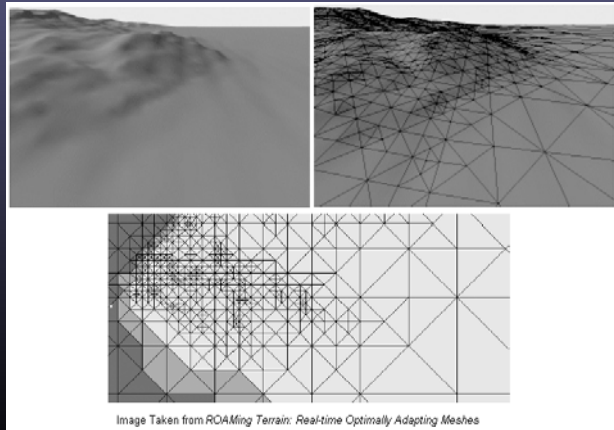
Reference

- A Method for Registration of 3-D Shapes
Paul J. Besl, Neil D. McKay
- Mode-based Decomposition of Part Form Error by Discrete-Cosine-Transform with Implementation to Assembly and Stamping System with Compliant Parts
Wenzhen Huang, Dariusz Ceglarek

Question?

ROAM

Real-time Optimally Adapting Meshes
Presented by: David Alegria



What is ROAM?

- Real-time Optimally Adapting Meshes
 - Main Purpose
 - Reduce the number of Polygons used to display terrains
 - Redistribute some processing from GPU to CPU
 - Created with Flight Simulators in Mind
 - Increased Frame Rates on very Large Terrains

Subdivisions of ROAM

- Triangle Bintree – Split and Merge
- Dual Queue Optimization
- Error Metrics
- Performance Enhancements

Triangle Bintree

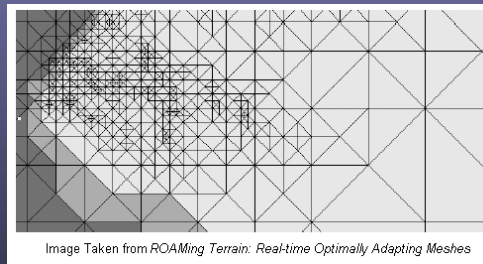
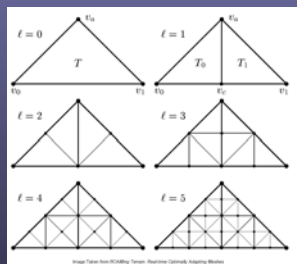


Image Taken from *ROAMing Terrain: Real-time Optimally Adapting Meshes*

Split and Merge

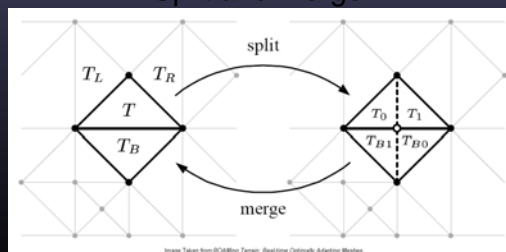


Image Taken from *ROAMing Terrain: Real-time Optimally Adapting Meshes*

Dual Queue Optimization

- Use two Priority Queues to expedite the split and merge process
 - Split Queue
 - Split Triangle based on Priority
 - Merge Queue
 - Helps with Improved performance from one frame to the next

Split Queue Sudo Code

```
Let T = the base triangulation
For All t elements of T, insert t int Split_Q
While T is too small or inaccurate
{
    Identify highest-priority t in Split_Q
    Force-split t
    Remove t and other split triangles from Split_Q
    Add any new triangles in T to Split_Q
}
```

Merge Queue Sudo Code

```
IF f = 0{
  let T = the base triangulation
  Clear Split_Q and Merge_Q
  Computer priorityies for elements of T
  Insert Triangles into Split_Q
  Insert diamonds into Merge_Q
}
else{
  Continue processing T = Tf-1
  Update Prioroties for All elements of Split_Q and Merge_Q
}
While (T is not the Target Size/accuracy, or the Maximum split Prioity is greter than
the minimum merge priority){
```

Merge Queue Sudo Code (continued)

```
if T is too large or accurate {
  Identify lowest-priority (T,Tb) in Merge_Q
  Merge (T,Tb)
  Remove All Merged childern from Split_Q
  Add merge parents T,Tb to Split_Q
  Remove(T,Tb) from Merge_Q
  Add all new mergable diamonds to Merge_Q
}
else{
  Identify highest priority T im Split_Q
  Force-split T
  Remove from Merge_Q any diamonds whose childern were split
  Add new Mergable Diamonds to Merge_Q
}
}
```

Error Metrics

- Nested World-Space Bounds
- Geometric Screen Distortion
- Line of Site Correction
- Others

Performance Enhancements

- View-Frustum Culling
- Incremental T-Stripping
- Deferring Priority Recomputation
- Progressive Optimization

Reference

Mark Dunchaineau, Murry Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstien, ROAMing Terrains: Real-time Optimally Adapting Meshes , 1997

Solar System Simulation

CIS 454
Computer Graphics

Setting Up

- `float CurrnetPlanetRotations[9] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };` //start rotaion at zero degrees
- `float PlanetRotationIncrement[9] = { 0.0f, 0.1f, 0.09f, 0.08f, 0.07f, 0.06f, 0.05f, 0.04f, 0.03f };` //rotation amount per frame
- `float PlanetRadius[9] = { 1.5f, 0.5f, 0.6f, 0.7f, 0.85f, 0.85f, 0.75f, 0.8f, 0.79f };` // size of planets
- `float PlanetDistance[9] = { 0, 1, 2, 3, 4, 6, 8, 11, 14 };` // width of the rings

Coloring The Planets

- `float PlanetColor[9][4] = {{1.0, 1.0, 0.0, 1.0},`
- `{0.8, 0.8, 0.8, 1.0},`
- `{1.0, 0.8, 0.0, 1.0},`
- `{0.2, 0.2, 1.0, 1.0},`
- `{1.0, 0.0, 0.0, 1.0},`
- `{0.8, 0.4, 0.0, 1.0},`
- `{0.5, 0.2, 0.0, 1.0},`
- `{0.0, 0.0, 0.6, 1.0},`
- `{0.4, 1.0, 1.0, 1.0}};`

Function to generate a sphere

- `glutSolidSphere(1.0f, 10, 10);`

Rotation and Rendering

- `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`//rotates camera based on mouse position
- `glRotatef(positionz, 1.0f, 0.0f, 0.0f);`//rotates the camera

Animates The Planets

- `for (i=0; i<9; ++i)`
- `{`
- `float x=0.0f,y = 0.0f;`
- `CurrtNetPlanetRotations[i] += PlanetRotationIncrement[i]/100.0f;`
- `x = PlanetDistance[i] * cos(CurrtNetPlanetRotations[i])*2.5f;`
- `y = PlanetDistance[i] * sin(CurrtNetPlanetRotations[i])*2.5f;`
- `glColor4fv(PlanetColor[i]);`
- `DrawSphere(x, 0.0f, y, PlanetRadius[i]);`
- `}`