Welcome to

# Scene Modeling Tools in Open Inventor

Demo Course

Sponsored by

Portable Graphics Inc

## Abstract

Open Inventor is a high-level cross-platform object-oriented 3-D interactive graphics and animation toolkit. This course covers necessary knowledge for creating and organizing scenes and objects in Open Inventor, including scene graph organization, shapes, properties, groups, lights, cameras, textures, Windows interfacing, VRML, and release 2.1 extensions. Basic concepts will be anchored with demonstrations programs which execute on one screen whild attendees examine associated source code on another.

PORTABLE GRAPHICS

## Speakers


- Chris Buckalew
  - buckalew@calpoly.edu (805) 756-1392
  - Computer Science Dept, Cal Poly State University
    San Luis Obispo, CA 93407
- John Readey
  - jlr@portable.com (512) 719-8000
  - Portable Graphics Inc, 3006 Longhorn Blvd Suite 105
    Austin, TX 78758
- Lew Hitchner
  - hitchner@phoenix.calpoly.edu (805) 756-2824
  - Computer Science Dept, Cal Poly State University
    San Luis Obispo, CA 93407


PORTABLE GRAPHICS

# Speaker Info

- **Chris Buckalew, Associate Professor**

  Chris Buckalew is an Associate Professor of Computer Science at Cal Poly State University in San Luis Obispo. H received his Ph.D. in 1990 from the University of Texas. His research interests include photorealistic image synthesis and scientific visualization.

  Dr. Buckalew's dissertation work was published in SIGGRAPH '89, and he ahs also published several articles on realistic image synthesis, scientific visualization, and computer-assisted lecture systems. He is currently engaged in building the undergraduate Computer Graphics program at Cal Poly, for which work he has received five consecutive annucal Outstanding Professor awards, voted on by the students.

- **John Readey, Product Manager**

  John Readey is the Open Inventor Product Manager at Portable Graphics Inc. He graduated from Ohio State University in 1989 with a M.S. degree in Computer Science. He spent the next five years at IBM where he developed IrisGl, and OpenGL software for the RS/6000. Since moving over to PGI in 1994, he has been engaged in Open Inventor porting issues, Inventor extensions, and VRML.

- **Lewis E. Hitchner, Lecturer**

  Dr. Hitchner obtained the Ph.D. degree from the University of Utah where he did research in 3D digital image processing and computer graphics. He was a faculty member in Computer Science at UC Santa Cruz for five years, and he is currently a lecturer in the Computer Science Department at California Polytechnic State University. His research and industrial employment includes four years Virtual Reality research at NASA Ames Research Center, two years in R&D for Xtensory Inc., and Sterling Software, Inc., and VR software development consulting for Sense8 Corp. Recently he has designed and taught technical training courses in VR software for Sense8 Corp. He is also the editor and author of 'The Virtual Software Report' published by the VR NEWS of London, UK.

PORTABLE GRAPHICS

## Table of Contents



PORTABLE GRAPHICS

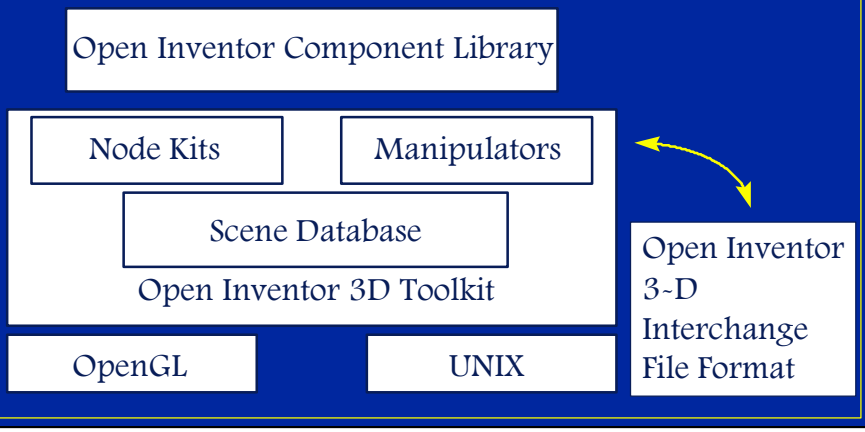
## Participant Background

- Knowledge of C or C++
- Basic computer graphics knowledge
- No graphics package or toolkit experience necessary



# What is Open Inventor?

- Object-oriented 3-D interactive graphics toolkit
- Library of objects and methods to create interactive 3-D graphics applications
- Standard across many vendors and platforms






## Object Database

- Extensible variety of primitives
- Objects can be picked, highlighted, and manipulated
- Object calculations such as bounding boxes and intersections may be performed
- Objects may be printed, searched for, rendered, and read to and from files



# Animation and Interaction

- Animation
  - Sensors
  - Field connections
  - Engines
- Interaction
  - Sensors
  - Callbacks
  - Selection
  - Draggers
  - Manipulators



## Open Inventor and OpenGL

- OpenGL is low-level to take direct advantage of graphics hardware
- Open Inventor is higher-level for ease of use, sophisticated applications, and speed of construction
- Open Inventor renders scenes with OpenGL calls
- Open Inventor includes much extra interaction and animation functionality as well as the scene graph conceptual framework
- OpenGL is assembly language; Open Inventor is a high-level language :-)



# Open Inventor and VRML

- The 2.1 release of Open Inventor added new classes useful for creating VRML applications:
  - SoWWWAnchor
  - SoWWWInline
  - SoAsciiText
  - SoFontStyle
- The VRML 1.0 file format is supported



## Rendering Capabilities

- Shapes:
  - Sphere, cone, cube, cylinder
  - Polyhedra
  - Text and 3-D text
  - NURBS curves and surfaces
  - Extensible to user-defined primitives
- Texture
- Transparency
- Access to all OpenGL rendering capabilities



## Sensors

- Timing sensors
  - Automatic triggering of timed events
- Data sensors
  - Activate callback procedures when data changes



## Engines

- Simple engine: the field connection
  - as one field changes, other fields hooked to it automatically change
- Most engines involve some function between connected fields
  - Animation engines: real-time clock drives engines to automatically update fields over time
  - Arithmetic engines: inputs from selected fields are to produce outputs that drive other fields
- Extremely flexible
- Encapsulates motion into metafile format



## Draggers and Manipulators

- Scene geometry that has built-in user interface and resulting actions
- Dragger output may be connected to any field for variety of applications
- Manipulators allow interactive editing of certain nodes



## Nodekits

- Organize nodes into subgraphs, (like functions in computer languages)
- Nodes are laid out in an efficient manner
- Resulting code is shorter and easier to understand
- Nodekits may be subclassed to create your own nodekit types



## Event Handling

- Automatic event handling
  - Selection node and manipulators
- Callbacks triggered by specific events
- Bypass Inventor event handling and receive events from the window system directly
- Callback nodes can trigger events during scene graph traversal



## File Format

- Stores scene geometry, engine motion, and automatic event handling
- Frequently faster to edit IV files rather than edit, recompile, and run programs
- File format is used for cutting and pasting between windows or processes
- Also used to specify nodekit parts
- Many converters are available



## Component Library

- Contains functions to communicate with the windowing system
- Includes variety of ***viewers*** and ***editors***
- Utility functions to manage windows
- Functions to customize the windows with toolbars, buttons, and menus
- Originally X-Windows; currently mature ports for Windows95 and NT with more planned



PORTABLE GRAPHICS

## Extensibility

- Create new shapes and nodekits
- Create property nodes that change current traversal state
- Create new group nodes that change order of traversal
- Create new traversal action
- Create new engines
- Create new draggers and manipulators
- Create new components
- Create new events and devices



# Starting Out

presented by

Chris Buckalew



PORTABLE GRAPHICS

# A Complete Program

```
#include <Inventor/Xt/SoXt.h>
#include<Inventor/Xt/SoXtRenderArea.h>
#include<Inventor/nodes/SoCone.h>
#include<Inventor/nodes/SoDirectionalLight.h>
#include<Inventor/nodes/SoMaterial.h>
#include<Inventor/nodes/SoPerspectiveCamera.h>
#include<Inventor/nodes/SoSeparator.h>

main(int argc, char **argv) {
  Widget window = SoXt::init(argv[0]);
  if (window == NULL) exit(1);

  //Make a scene containing a red cone
  SoSeparator *root = new SoSeparator;
  SoPerspectiveCamera  *camera = new SoPerspectiveCamera;
  SoMaterial  *material = new SoMaterial;
```

Starting Out

intro1.C (from Inventor Mentor)

PORTABLE GRAPHICS

# A Complete Program (continued)

```
    root->ref( );
    root->addChild(camera);
    root->addChild(new SoDirectionalLight);
    material->diffuseColor.setValue(1.0, 0.3, 0.3);
    root->addChild(material);
    root->addChild(new SoCone);

    SoXtRenderArea  *rarea = new SoXtRenderArea;
    camera->viewAll(root, rarea->getViewportRegion( ));

    rarea->setSceneGraph(root);
    rarea->show( );

    SoXt::show(window);
    SoXt::mainLoop( );
}
```

Starting Out

intro1.C (from Inventor Mentor)

PORTABLE GRAPHICS

# Make the Cone Spin

```
#include <Inventor/engines/SoElapsedTime.h>
#include <Inventor/nodes/SoRotationXYZ.h>

main(int  argc,  char  **argv) {
...
   SoRotationXYZ  *rotate = new RotationXYZ;
   root->addChild(rotate);
...
   rotate->axis = SoRotationXYZ::X;
   SoElapsedTime  *counter = new SoElapsedTime;
   rotate->angle.connectFrom(&counter->timeOut);
...
}
```

Starting Out

intro2.C (from Inventor Mentor)

PORTABLE GRAPHICS

# Adding a Manipulator

```
#include <Inventor/engines/SoTrackballManip.h>

main(int  argc,  char  **argv) {
...
   root->addChild(new SoTrackballManip);
...
}
```

Starting Out

intro3.C (from Inventor Mentor)

PORTABLE GRAPHICS

# Viewing with the ExaminerViewer

```
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>

main(int  argc,  char  **argv) {
...
  SoXtExaminerViewer  *viewer =
          new SoXtExaminerViewer(window);
...
}
```

Starting Out

intro4.C (from Inventor Mentor)

PORTABLE GRAPHICS

## Naming Conventions

- Basic types begin with Sb (Scene basic):
  - SbVec3f, SbColor
- Other classes begin with So (Scene object):
  - SoCone, SoMaterial, SoTransform
- Methods, variables, and fields begin with lowercase; included words are capitalized:
  - setValue(...), firstCube, .diffuseColor
- Enumerated types are all uppercase:
  - FILLED, X, COUNTERCLOCKWISE

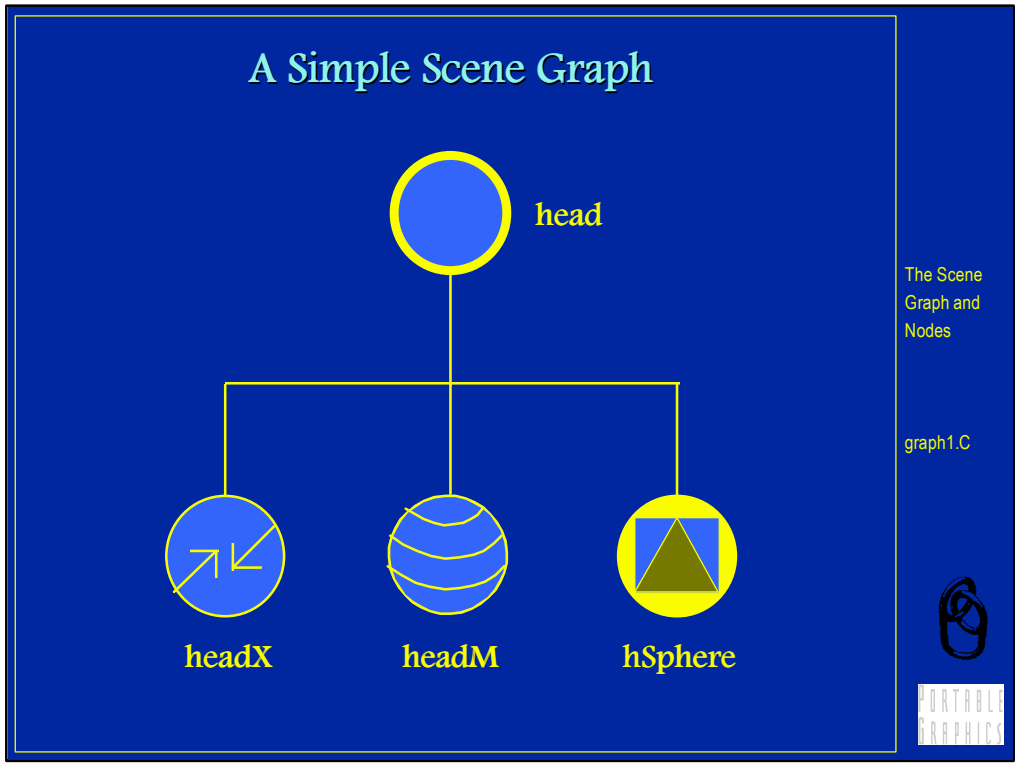


# The Scene Graph

presented by

Chris Buckalew




A Simple Scene Graph
head
headX
headM
hSphere
The Scene Graph and Nodes
graph1.C
PORTABLE GRAPHICS

## Types of Nodes

- Shape nodes
  - rendered using current state when traversed
- Property nodes
  - change the current state
- Group nodes
  - govern order of traversal
  - some group nodes save and restore the current state



# Creating Nodes

```
// make a head
SoSphere  *hSphere = new SoSphere;

//stretch it
SoTransform  *headX = new SoTransform;
headX->scaleFactor.setValue(5.0, 7.0, 5.0);

//color it
SoMaterial  *headM = new SoMaterial;
headM->diffuseColor.setValue(0.9, 0.7, 0.6);
```



graph1.C

# Building a Scene Graph

```
// make the root
SoGroup  *head = new SoGroup;

// other nodes have been created already

head->addChild(headX);
head->addChild(headM);
head->addChild(hSphere);
```

The Scene Graph and Nodes

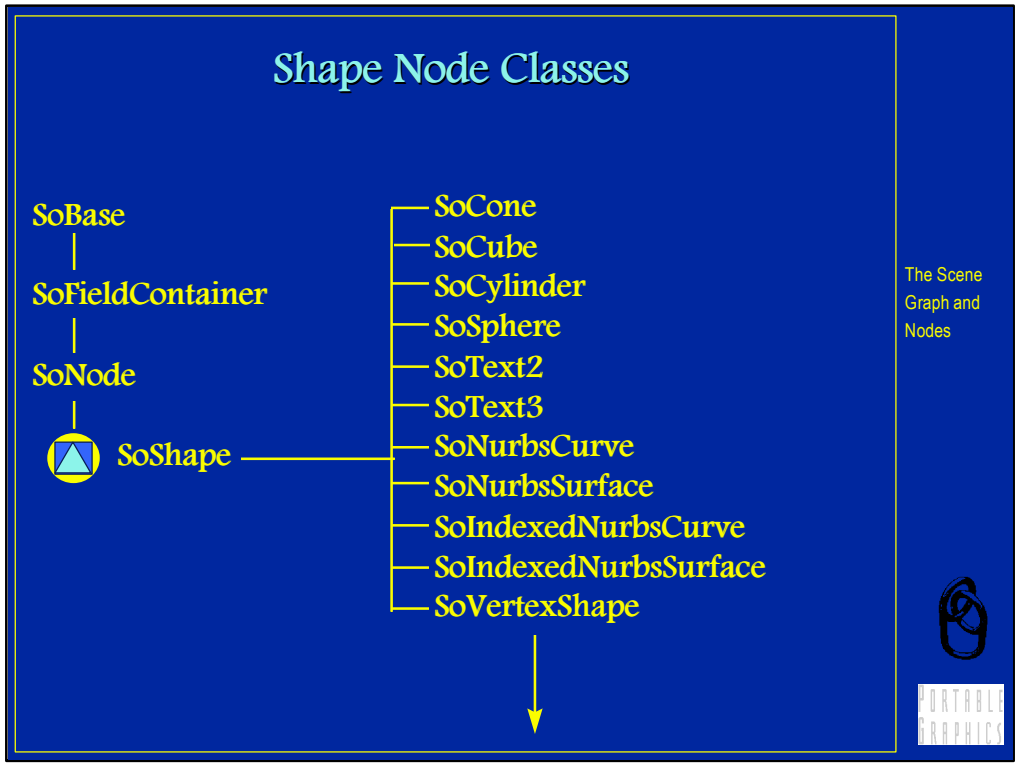
graph1.C

PORTABLE GRAPHICS

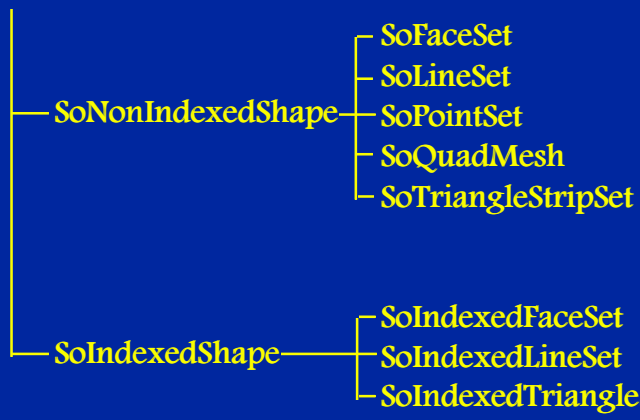
# Node Fields

- SoSphere:
  - radius
- SoTransform:
  - translation
  - rotation
  - scaleFactor
  - scaleOrientation
  - center
- SoMaterial:
  - ambientColor
  - diffuseColor
  - specularColor
  - emissiveColor
  - shininess
  - transparency



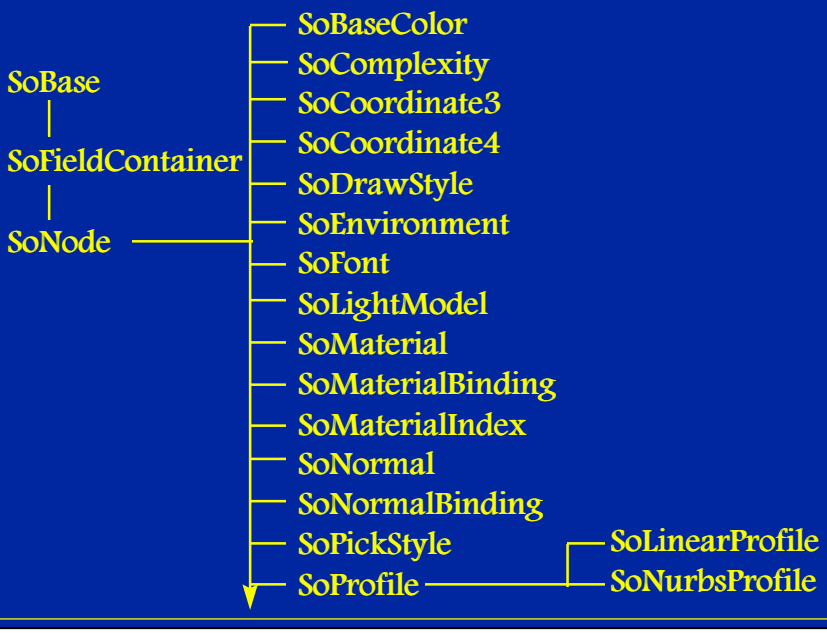

Shape Node Classes
SoBase
SoFieldContainer
SoNode
SoShape
SoCone
SoCube
SoCylinder
SoSphere
SoText2
SoText3
SoNurbsCurve
SoNurbsSurface
SoIndexedNurbsCurve
SoIndexedNurbsSurface
SoVertexShape
The Scene Graph and Nodes
PORTABLE GRAPHICS

# Shape Node Classes (cont)






# Property Node Classes



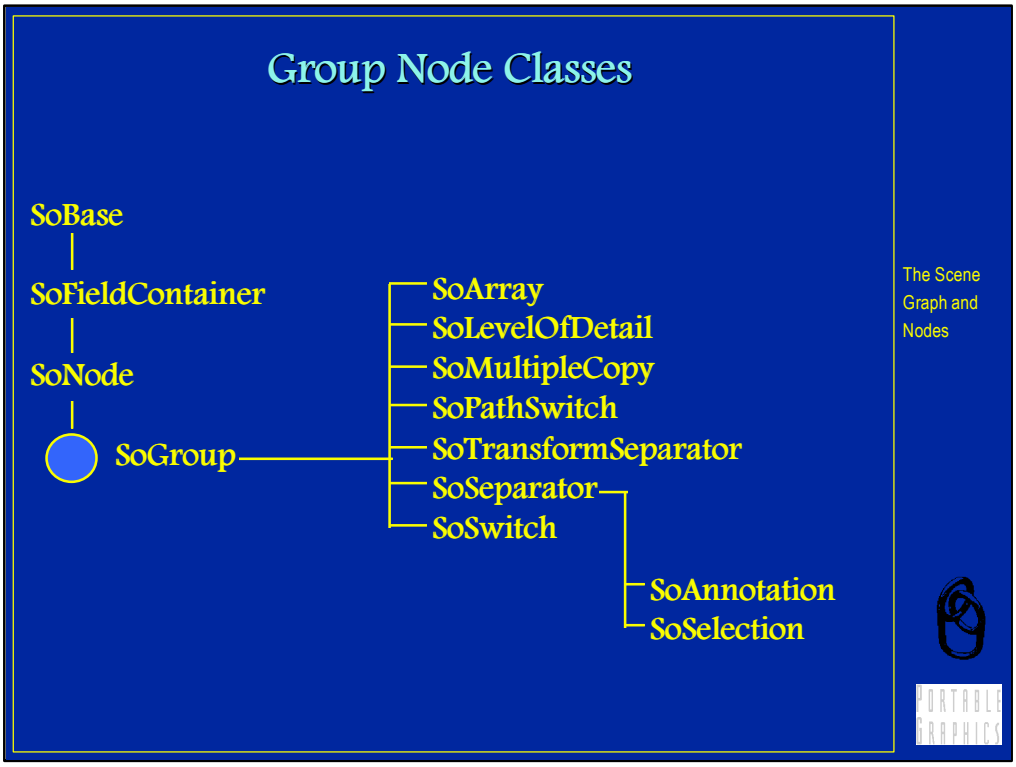


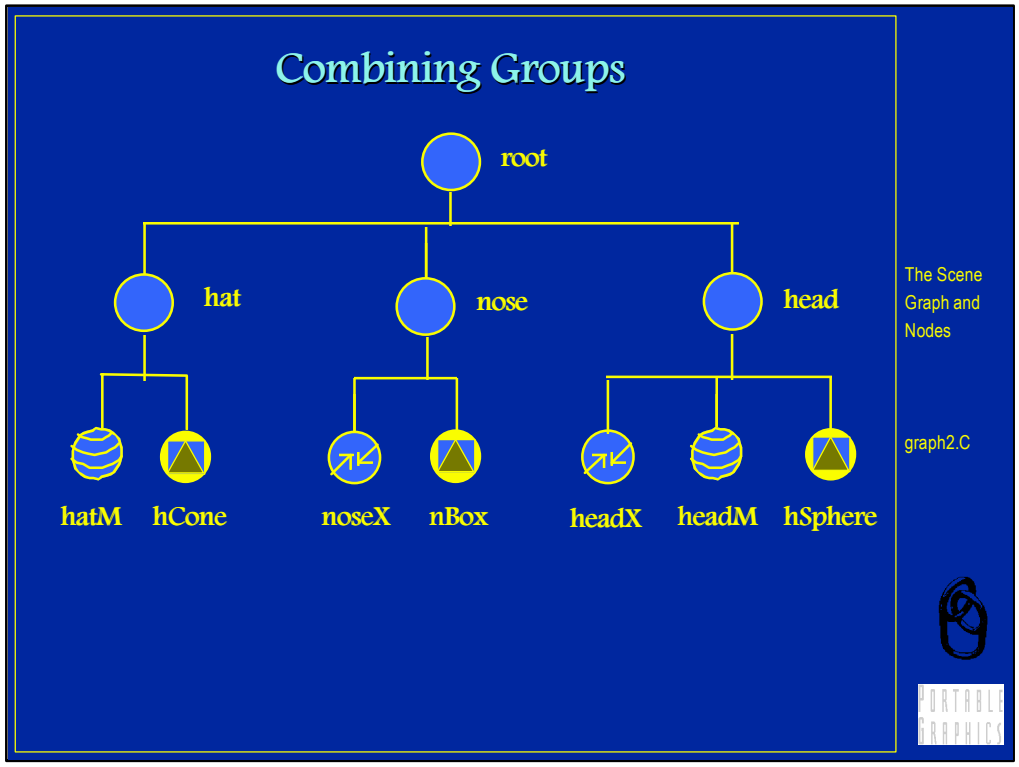

# Property Node Classes (cont)

- SoProfileCoordinate2
- SoProfileCoordinate3
- SoShapeHints
- SoTexture2
- SoTexture2Transform
- SoTextureCoordinate2
- SoTextureCoordinateBinding
- SoTextureCoordinateFunction
- SoTransformation
  - SoAntiSquish
  - SoMatrixTransform
  - SoResetTransform
  - SoRotation
  - SoRotationXYZ
  - SoScale
  - SoSurroundScale
  - SoTransform
  - SoTranslation
  - SoUnits
- SoUnknownNode



# Group Node Classes



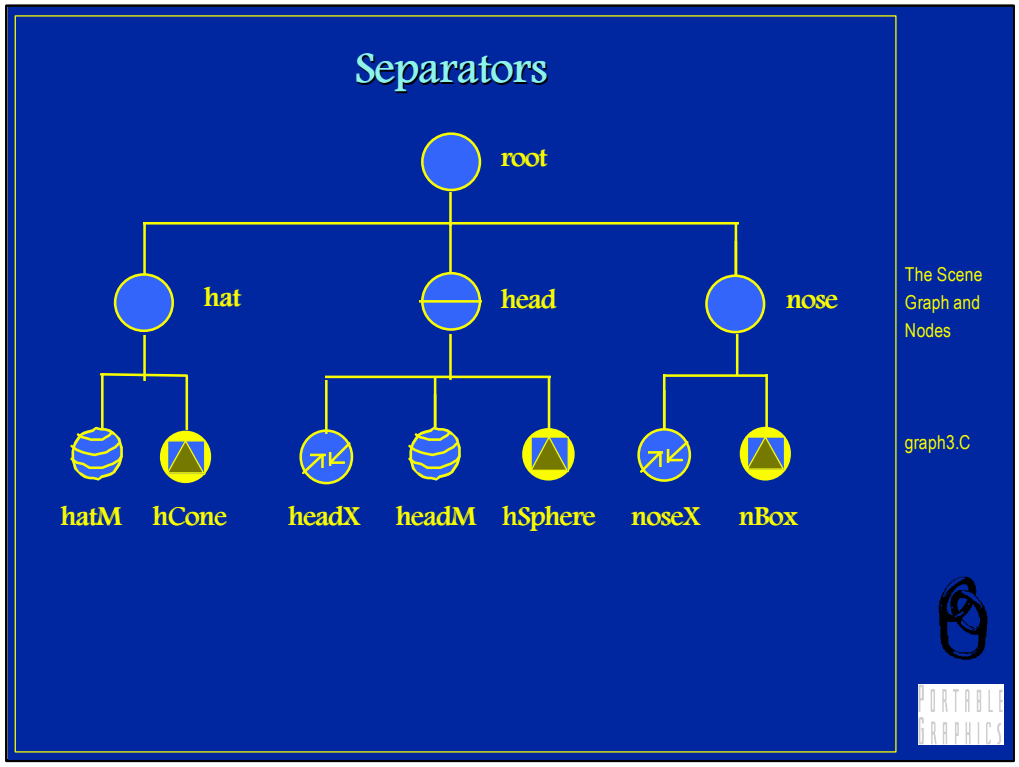

Combining Groups
root
hat
nose
head
hatM
hCone
noseX
nBox
headX
headM
hSphere
The Scene Graph and Nodes
graph2.C
PORTABLE GRAPHICS

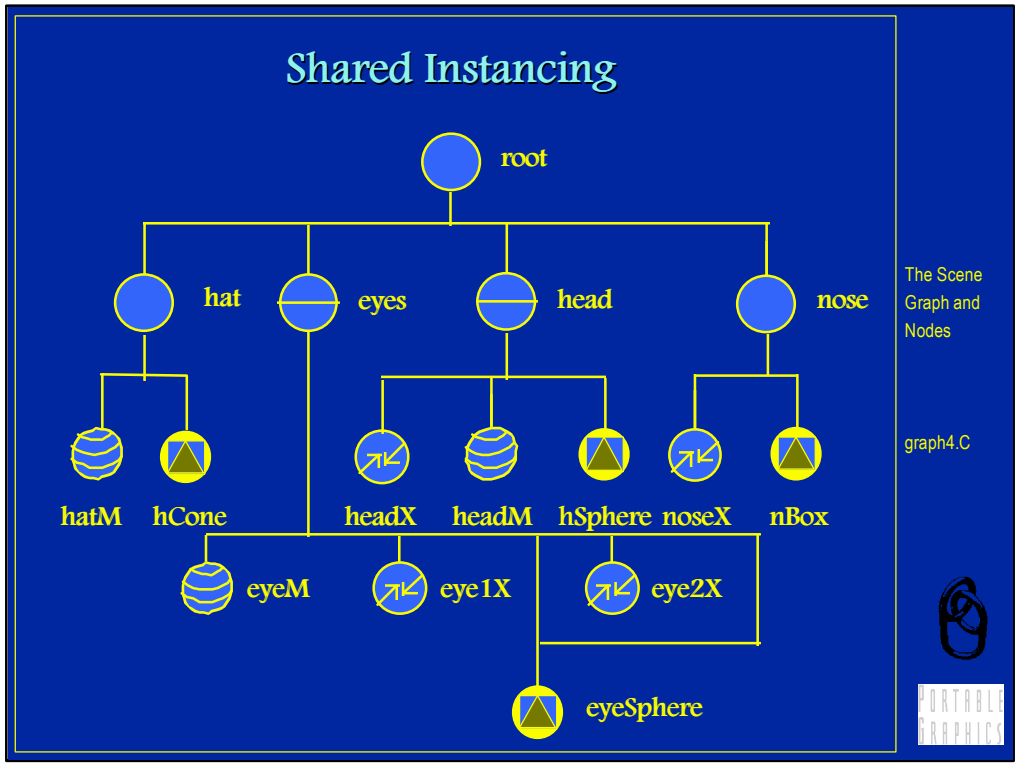
# Combining Groups Example

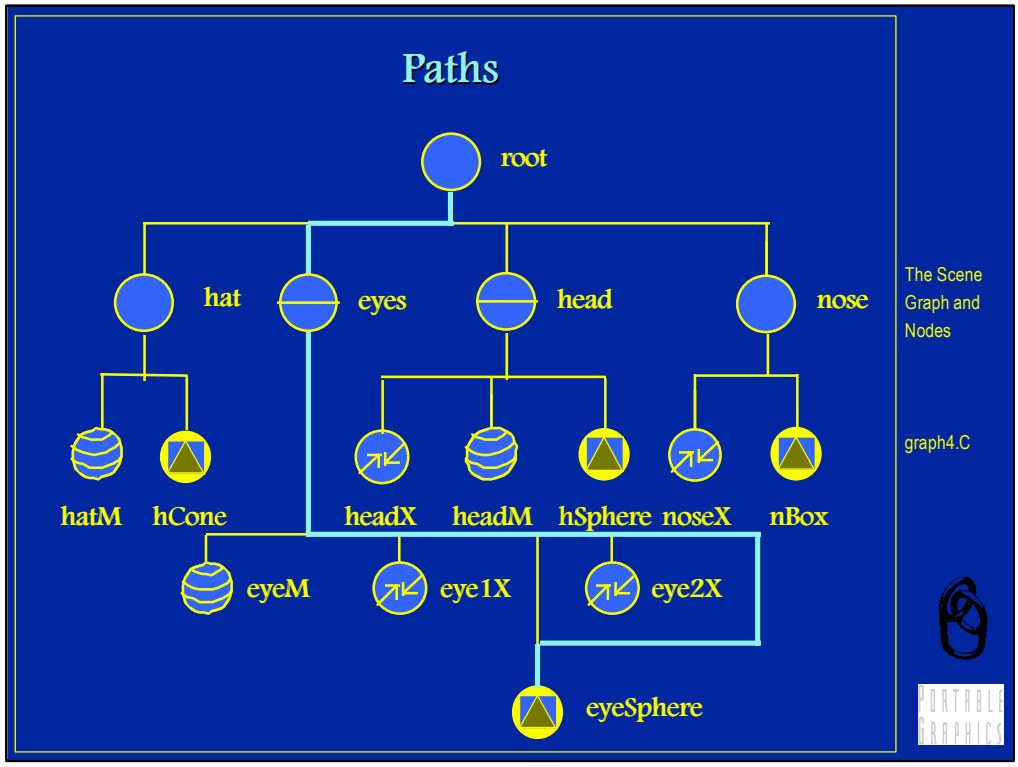
```
SoGroup  *root = new SoGroup;
   SoGroup  *hat = new SoGroup;
      hat->addChild(hatM);
      hat->addChild(hCone);
   root->addChild(hat);

   SoGroup  *nose = new SoGroup;
      nose->addChild(noseX)
      nose->addChild(nBox);
   root->addChild(nose);

   SoGroup  *head = new SoGroup;
      head->addChild(headX);
      head->addChild(headM);
      head->addChild(hSphere);
   root->addChild(head);
```



graph2.C


Separators
root
hat
head
nose
hatM
hCone
headX
headM
hSphere
noseX
nBox
The Scene Graph and Nodes
graph3.C
PORTABLE GRAPHICS


Shared Instancing
root
hat
eyes
head
nose
hatM
hCone
headX
headM
hSphere
noseX
nBox
eyeM
eye1X
eye2X
eyeSphere
The Scene Graph and Nodes
graph4.C
PORTABLE GRAPHICS


Paths
root
hat
eyes
head
nose
hatM
hCone
headX
headM
hSphere
noseX
nBox
eyeM
eye1X
eye2X
eyeSphere
The Scene Graph and Nodes
graph4.C
PORTABLE GRAPHICS

## Fields within a Node

- Floats, Longs, and Shorts

```
SoCube  *box = new SoCube;

// setting fields
box->width = 4.0;
box->height.setValue(6.5);

// getting fields
SoSFFloat  size = box->width;
float  distance = box->height.getValue;
```



# Fields within a Node

- Vectors

```
SoTransform  *Xform = new SoTransform;

// set from a vector
SbVec3f  vector;
vector.setValue(1.5, 3.2, 4.3);
Xform->translation.setValue(vector);
Xform->translation = vector;
Xform->translation.setValue(SbVec3f(1.5, 3.2, 4.3);

// set from three floats
Xform->translation.setValue(1.5, 3.2, 4.3);

// set from an array of floats
float  array[3] = {1.5, 3.2, 4.3};
Xform->translation.setValue(array);
```



# Fields within a Node

- Rotations

```
SbRotation  rot;
SbVec3f  axisX(1.0, 0.0, 0.0);
SbVec3f  axisY(0.0, 1.0, 0.0);
float  angle = M_PI;

// using the setValue method
rot.setValue(axisX, angle);
rot.setValue(axisX, axisY);

   // using the constructor
SbRotation  rot1(SbVec3f(1.0, 0.0, 0.0),  M_PI);

   // rotation one vector into another
SbRotation  rot2(SbVec3f(1.0, 0.0, 0.0), SbVec3f(0.0, 1.0, 0.0));
```



# Lights and Cameras

presented by

Chris Buckalew



# Cameras

- useful SoCamera fields
  - position
  - orientation
  - nearDistance
  - farDistance
  - focalDistance
  - aspectRatio
- useful SoCamera methods
  - pointAt()
  - viewAll()



cameras.C

# Camera Types

- SoPerspectiveCamera
  - heightAngle field determines field-of-view
- SoOrthographicCamera
  - height field determines width-of-field



# Lights

- A light illuminates nodes that follow it in the scene graph
- Light positions and orientations are affected by the current geometric transformation
- Lights accumulate



## Light Fields

- SoLight
  - on
  - intensity
  - color
- SoPointLight
  - location
- SoDirectionalLight
  - direction
- SoSpotLight
  - location
  - direction
  - dropOffRate
  - cutOffAngle



lights.C

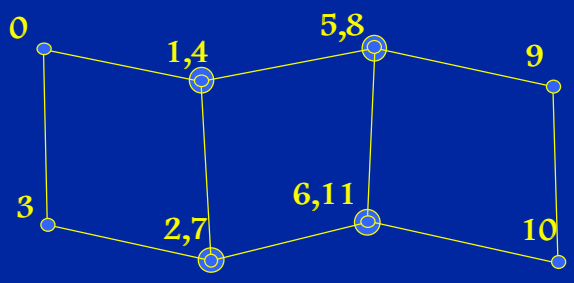
# Building Objects

presented by

Lew Hitchner



# Face Set

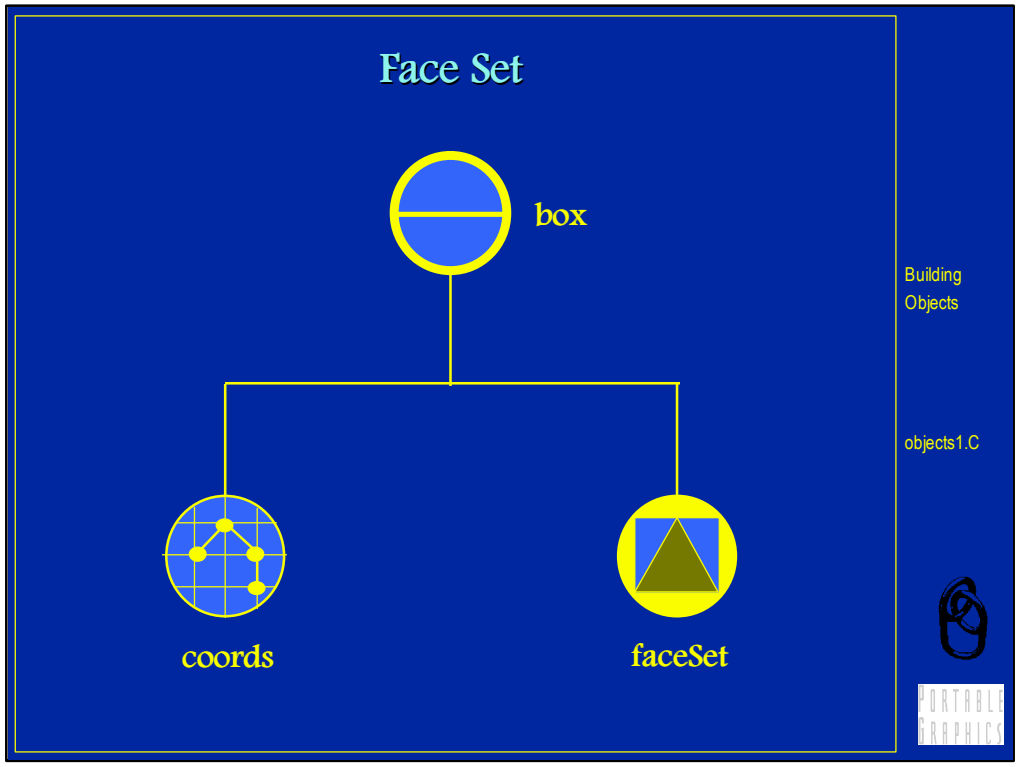



vertices[12][3]

SoFaceSet:

numVertices: {4, 4, 4}




Face Set
box
coords
faceSet
Building
Objects
objects1.C
PORTABLE
GRAPHICS

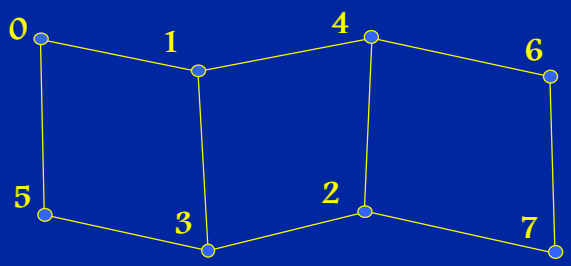
# Face Set Example

```
SoSeparator *makeEars( ) {
   float  earVerts[24][3] = {{1.0, 0.0, 0.0},...};
   long numEarVerts[8] = {3,3,3,3,3,3,3,3};

   SoCoordinate3  *earCoords = new SoCoordinate3;
   earCoords->point.setValues(0, 24, earVerts);
   ears->addChild(earCoords);

   SoFaceSet  *faceSet = new SoFaceSet;
   faceSet->numVertices.setValues(0, 8, numEarVerts);
   ears->addChild(faceSet);

   return  ears;
}
```



objects1.C

# Indexed Face Set




vertices[8][3]

indices[15] = {0, 1, 3, 5, SO_END_FACE_INDEX,...}

SoFaceSet:

coordIndex: 15



# Indexed Face Set Example

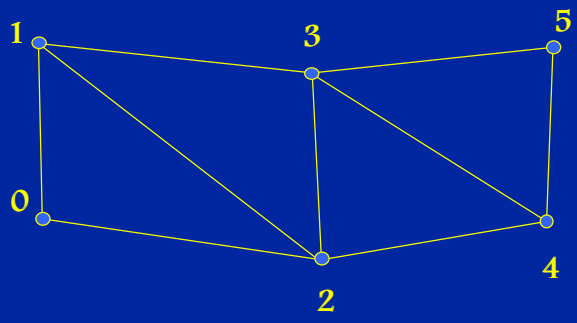
```
SoSeparator *makeEars( ) {
   float earVerts[9][3] = {{1.0, 0.0, 0.0},...};
   int earIndex[32] = {0, 1, 2, -1, ...};

   SoCoordinate3 *earCoords = new SoCoordinate3;
   earCoords->point.setValues(0, 9, earVerts);
   ears->addChild(earCoords);

   SoIndexedFaceSet *faceSet = new SoIndexedFaceSet;
   faceSet->coordIndex.setValues(0, 32, earIndex);
   ears->addChild(faceSet);

   return ears;
}
```

Building Objects

objects2.C

PORTABLE GRAPHICS

# Triangle Strip Set




vertices[6][3]

SoTriangleStripSet:
numVertices: 6



# Triangle Strip Example

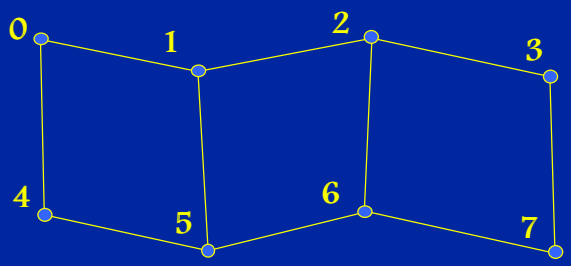
```
SoSeparator *buildSmile( ) {
   float  smileVerts[28][3] = {{1.0, 0.8, 1.0}, ...}
   int smileIndex[3] = {8, 10, 10};

   SoCoordinate3  *smileCoords = new SoCoordinate3;
   smileCoords->point.setValues(0, 28, smileVerts);
   smile->addChild(smileCoords);

   SoTriangleStripSet  *mesh = new SoTriangleStripSet;
   mesh->numVertices.setValues(0, 3, smileIndex);
   smile->addChild(mesh);

   return  smile;
```

Building Objects

objects3.C

PORTABLE GRAPHICS

# Quad Mesh




vertices[8][3]

SoQuadMesh:

verticesPerRow: 4

verticesPerColumn: 2



# Quad Mesh Example

```
SoSeparator *buildRuff( ) {
   float ruffVerts[34][3] = {{0.0, 2.0, 1.0}, ...};

   SoSeparator *ruff = new SoSeparator;

   SoCoordinate3 *ruffCoords = new SoCoordinate3;
   ruffCoords->point.setValues(0, 34, ruffVerts);
   ruff->addChild(ruffCoords);

   SoQuadMesh *quadMesh = new SoQuadMesh;
   quadMesh->verticesPerRow = 17;
   quadMesh->verticesPerColumn = 2;
   ruff->addChild(quadMesh);
   return ruff; }
```

Building Objects

objects4.C

PORTABLE GRAPHICS

# Order of Transformations

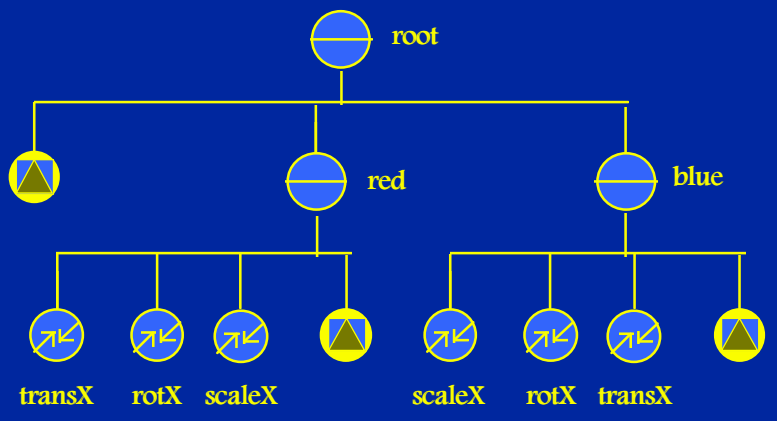
- Within the SoTransformation node, the fields are applied so that the last field affects the object first:

- translation
- rotation
- scaleFactor
- scaleOrientation
- center



# Accumulating Transformations

- In the scene graph, transform nodes closest to the object (and to the left) affect it first






## NURBS Curves and Surfaces

- Non-Uniform Rational B-Spline
- Includes B-Splines, Beziers, and others
- Curves in 3-Space
- Surfaces in 3-Space
- Trimmed surfaces



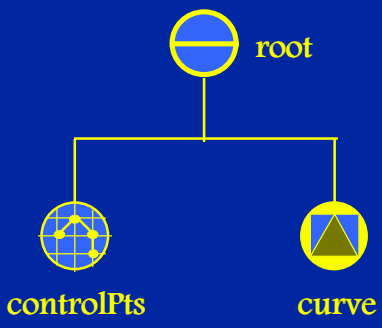
# Types of Curves

- Uniform cubic B-spline:
  - float pts[7][3] = {{-6.0, -6.0, 0.0}, ...};
  - int knots[11] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
- Cubic B-spline that interpolates endpoints:
  - float pts[7][3] = {{-6.0, -6.0, 0.0}, ...};
  - int knots[10] = {0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4};
- Cubic Bezier curve:
  - float pts[4][3] = {{-6.0, -6.0, 0.0}, ...};
  - int knots[8] = {0, 0, 0, 0, 1, 1, 1, 1};



nurbs1.C

# Curve Scene Graph




controlPts[7][3]

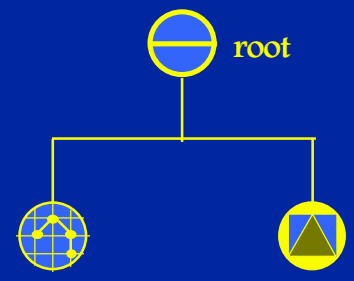
SoNurbsCurve:

numControlPoints: 7

knotVector: {0,0,0,0,1,1,1,1}



# Surface Scene Graph




controlPts[16][3]

SoNurbsSurface:

numUControlPoints: 4

numVControlPoints: 4

uKnotVector: {0,0,0,0,1,1,1,1}

vKnotVector: {0,0,0,0,1,1,1,1}



## Trimmed NURBS Surfaces

- Profile curves trim surfaces
- Profile curves are defined in parameter space
- Must form closed loop
- Clockwise loop cuts a hole in the surface; counterclockwise loop trims off edges
- Profile curves may be nested but cannot intersect themselves or each other
- Profile nodes are SoLinearProfile or SoNurbsProfile



nurbs3.C

# 2-D Text

- Screen-aligned 2-D text; does not vary in size with camera movement
- SoText2 fields:
  - string: text to display
  - spacing: distance between lines of text
  - justification: LEFT, CENTER, RIGHT
- SoFont fields:
  - name: font name
  - size: in points



## 2-D Text Example

```
SoSeparator *twoDText( ) {
   SoSeparator *text = new SoSeparator;

   SoFont *font = new SoFont;
   font->name.setValue("Times-Roman");
   font->size.setValue(24.0);
   text->addChild(font);

   SoText2 *word = newSoText2;
   word->string = "Open Inventor";
   text->addChild(word);
return text; }
```

Building Objects

text1.C

PORTABLE GRAPHICS

# 3-D Text

- 3-D text has depth and can be scaled; does not always remain parallel to the screen
- SoText3 fields:
  - string: text to display
  - spacing: distance between lines of text
  - justification: LEFT, CENTER, RIGHT
  - parts: FRONT, SIDES, BACK, ALL

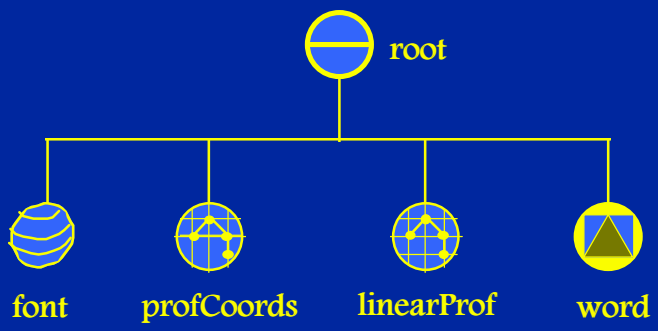
Building Objects

text2.C

PORTABLE GRAPHICS

# 3-D Text Profiles

- Defines the shape of the sides of the 3D text
- Uses SoLinearProfile or SoNurbsProfile




Building Objects

text3.C

PORTABLE GRAPHICS

# 3-D Text Example

```
const float pts[5][2]=
    {{0,0},{0.25,0.25},{0.5,0.15},{0.75,0.25},{1,0}};
const int indices[5] = {0, 1, 2, 3, 4};
...
SoProfileCoordinate2  *profCoords = new
   SoProfileCoordinate2;
profCoords->point.setValues(0, 5, pts);
text->addChild(profCoords);

SoLinearProfile  *linearProf = new SoLinearProfile;
linearProf->index.setValues(0, 5, indices);
...
```

Building Objects

text3.C

PORTABLE GRAPHICS

# Textures

presented by

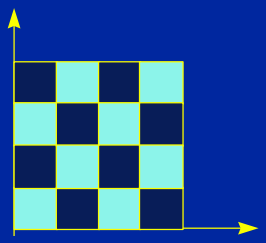
Lew Hitchner



PORTABLE GRAPHICS

# Textures

- A texture is a 2-D array of pixels in parameter space
- Texture pixels are mapped onto surfaces



Textures

texture1.C

## Scanned Texture Example

```
main( ) {
    ...
    // get texture from file
    SoTexture2  *postcard = new SoTexture2;
    root->addChild(postcard);
    postcard->filename.setValue("postcard.rgb");

    root->addChild(new SoCube);
    ...
}
```

Textures

texture2.C

PORTABLE GRAPHICS

## Texture Effects

- MODULATE: multiplies texture color and object's shaded color, including transparency
  - example: box color (0.7, 1.0, 0.3) modulated by texture colors (1, 1, 1) and (1, 0, 0)
- DECAL: replaces object's shaded color with texture color. Texture alpha component determines texture transparency
  - example: box color replaced by texture colors
- BLEND: texture intensity blends between object's shaded color and a constant blend color
  - example: box color blended by texture intensities 1 and 0 with blend color (0, 0, 1)

Textures

texture3.C

texture4.C

texture5.C

PORTABLE GRAPHICS

## Texture Components

- one-component: intensity only
- two-component: intensity and transparency
- three-component: RGB values
- four-component: RGB and transparency

- MODULATE is used with all types
- DECAL is used with 3- and 4-component textures
- BLEND is used with 1- and 2-component textures



# Building Textures from Scratch

Three-component texture:

```
SoTexture2  *texture = new SoTexture;

unsigned char image[ ] = {
    255, 255, 255,        0, 0, 0,
    0, 0, 0,              255, 255, 0};

texture->image.setValue(SbVec2s(2, 2), 3, image
```

| 6 | 7 | 8 |
|---|---|---|
| 0 | 1 | 2 |

| 9 | 10 | 11 |
|---|---|---|
| 3 | 4 | 5 |

Example shows a 4-component texture with transparency

Textures

texture6.C

PORTABLE GRAPHICS

# Transforming Textures

- Textures are transformed with the SoTexture2Transform
- SoTexture2Transform fields:
  - translation
  - rotation
  - scaleFactor
  - center: center of scale and rotation



# Transforming Texture Example

```
main( ) {
   ...
   SoTexture2  *texture = new SoTexture2;
   ...
   texture->image.setValue(SbVec2s(2,2), 3, image);

   SoTexture2Transform  *tranX =
                new SoTexture2Transform;
   transX->rotation.setValue(M_PI / 6.0);
   transX->scaleFactor.setValue(8.0, 8.0);

   root->addChild(transX);
   root->addChild(texture);
   root->addChild(new SoCube);
}
```

Textures

texture7.C

PORTABLE GRAPHICS

# Explicit Texture Coordinates

- Textures may be attached manually to vertices of objects
- For each object vertex, a corresponding texture coordinate pair must specified (in texture parameter space



texture8.C

# Explicit Texture Coordinates Example

```
SoCoordinate3  *coords = new SoCoordinate3;
coords->point.set1Value(0, SbVec3f( -2,  -2,   0  ));
coords->point.set1Value(1, SbVec3f(   2,  -2,   0  ));
coords->point.set1Value(2, SbVec3f(   2,   2,   0  ));
coords->point.set1Value(3, SbVec3f( -2,   2,   0  ));
…
SoTextureCoordinate2  *texCoord = new SoTextureCoord…
texCoord->point.set1Value(0, SbVec2f(  0.0,   0.0  ));
texCoord->point.set1Value(1, SbVec2f(  1.0,   0.0  ));
texCoord->point.set1Value(2, SbVec2f(  1.0,   0.5  ));
texCoord->point.set1Value(3, SbVec2f(  0.0,  0.25  ));
texBin->value.setValue(
                SoTextureCoordinateBinding::PER_VERTEX);
```

Textures

texture8.C

PORTABLE GRAPHICS

## Texture Coordinate Functions

- Allow you to specify texture coordinates implicitly rather than explicitly
- SoTextureCoordinatePlane: project the texture through a plane
- SoTextureCoordinateEnvironment: specifies a spherical environment map



## SoTextureCoordinatePlane

- fields:

  directionS: projection direction of *s* coordinate
  default is ( 1.0, 0.0, 0.0 )

  directionT: projection direction of *t* coordinate
  default is ( 0.0, 1.0, 0.0 )

- repeat interval is length of direction vector



texture9.C

## SoTextureCoordinatePlane Example

```
SoTextureCoordinatePlane  *texPlane1 = new …
root->addChild(texPlane1);
root->addChild(new SoSphere);

// translation here
SoTextureCoordinatePlane  *texPlane2 = new …
texPlane2->directionS.setValue(SbVec3f(1, 1, 0));
texPlane2->directionT.setValue(SbVec3f(0, 1, 1));
root->addChild(texPlane2);
root->addChild(new SoSphere);
```

Textures

texture9.C

texture10.C

PORTABLE GRAPHICS

## SoTextureCoordinateEnvironment

- Simulates a raytraced image by texture-mapping on an object a pattern that appears to be a reflection of the scene around the object
- Uses a spherical reflection map, which must be created separately
- Works well in static environments, but if camera moves or the environment near the object changes the reflection map must be recomputed and reapplied

Textures

texture11.C

PORTABLE GRAPHICS

# Recent Developments

Extensions
and
New Stuff

presented by

John Readey

PORTABLE GRAPHICS

## Resources

- *The Inventor Mentor* and *The Inventor Toolmaker*, by Josie Wernecke, Addison-Wesley (also on-line on SGI machines)
- *The Open Inventor C++ Reference Manual*, Addison-Wesley (on line as man pages)
- Web pages:
  - Silicon Graphics: www.sgi.com
  - Portable Graphics: www.portable.com
  - Template Graphics: www.