Welcome to

Programming Animation and Interaction 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 programming animation and interaction in Open Inventor, including sensors, engines, manipulators, events, Windows interfacing, and performance optimization. Basic concepts will be anchored with demonstration programs which execute on one screen while 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

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.

Table of Contents

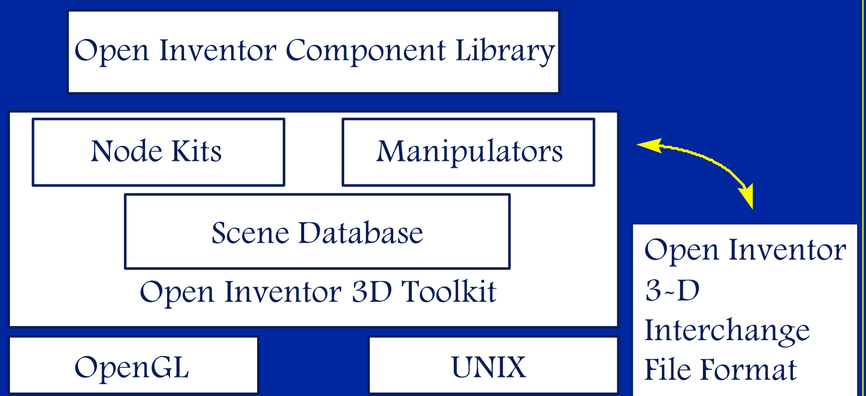
PORTABLE GRAPHICS

Participant Background

- Knowledge of C or C++
- Basic computer graphics knowledge
- Some knowledge of Open Inventor: nodes, scene graph organization, properties, traversal, etc.

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



PORTABLE GRAPHICS

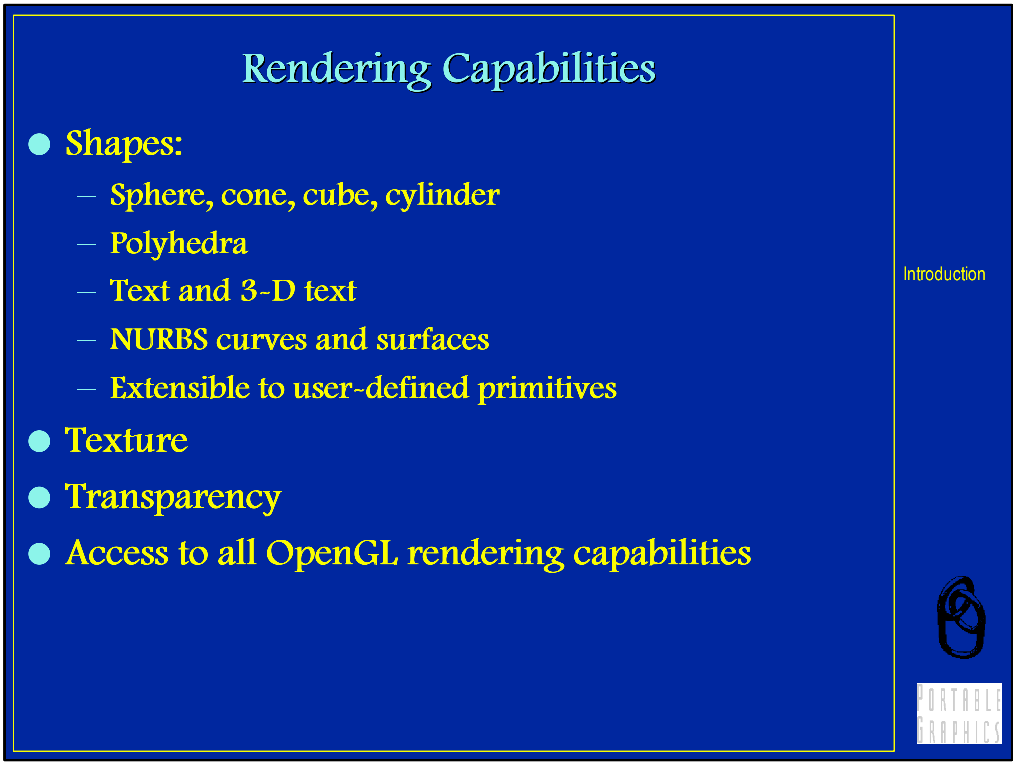
Object Database

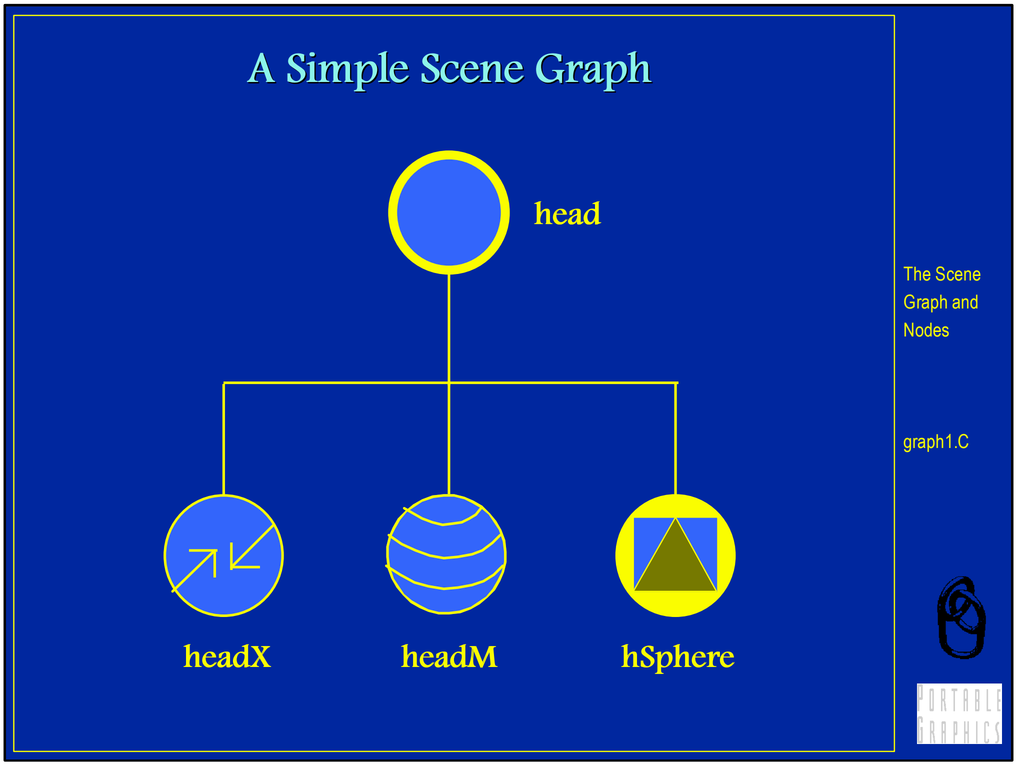
- Extensible variety of primitives
- Objects can be picked, highlighted, and manipulated
- Object calculations such as bounding boxes and intersections may be performed
- Objects may be printed, searched for, rendered, and read to and from files

Animation and Interaction

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

PORTABLE GRAPHICS


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
Introduction


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

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

Sensors

presented by

Chris Buckalew

Sensors

- Timer queue sensors
  - Alarm sensor
  - Timer sensor
- Delay queue sensors
  - Field, node, and path sensors
  - Idle sensor
  - Oneshot sensor

Sensor Callbacks

- Defining a callback:

```
static void
lightCallback(void *data, SoSensor *) {
   SoDirLight *light = (SoDirLight *) data;
   if (light->on.getValue( ) == TRUE)
      light->on = FALSE;
   else
      light->on = TRUE;
}
```


sensor1.C

Sensor Callbacks (cont)

- Initializing a callback function:

```
SoDirLight  *dLight = new SoDirLight;
SoTimerSensor  *lightSensor =
     new SoTimerSensor(lightCallback, dLight);
lightSensor->setInterval(5.0);
lightSensor->schedule( );
```


sensor1.C

SoAlarmSensor Example

```
static void alarmCallback(void *data, SoSensor *) {
    SoDirLight *light = (SoDirLight *) data;
    light->on = TRUE; }

main( ) {
    SoDirLight *dLight = new SoDirLight;
    SoAlarmSensor *alarm =
            new SoAlarmSensor(alarmCallback, dLight);
    alarm->setTimeFromNow(10.0);
    alarm->schedule( );
}
```

Data Sensors

- Sensor is attached to a field, node, or path
- Data sensors have priorities
- When the data changes:
  - the sensor is scheduled according to priority
  - later, the delay queue is processed and sensor is triggered
  - triggered sensor is removed from queue and callback is executed

Field Sensor Example

```
cameraLocCB(void  *data, SoSensor  *) {
   SoAlarmSensor  *delay = (SoAlarmSensor  *) data;
   drawStyle->style = SoDrawStyle::LINES;
   delay->unschedule( );
   delay->setTimeFromNow(3.0);
   delay->schedule( );    }

main( )  {
   SoAlarmSensor  *delay =
         new SoAlarmSensor(restoreFillCB, NULL);
   SoCamera  *camera = viewer->getCamera( );
   SoFieldSensor  *fieldSensor =
         new SoFieldSensor(cameraLocCB, delay);
   fieldSensor->attach(&camera->position);
}
```

Sensors

sensor3.C

Trigger Fields

- Use trigger methods to find which field triggered the callback
- Within the callback function:

```
SoField  *changed = sensor->getTriggerField( );
```

- Useful when several sensors execute the same callback function
- Trigger node finds which node triggered callback

sensor4.C

One-Shot and Idle Sensors

- SoOneShotSensor is triggered when the delay queue is processed
  - use to delay time-consuming work
  - guaranteed to execute periodically
- SoIdleSensor is triggered when there are no events or timers to be processed
  - may never be triggered if CPU stays busy
- Delay queue is processed every 1/30th second by default
  - interval may be changed with

    SoDB::setDelaySensorTimeout( )

PORTABLE GRAPHICS

Engines

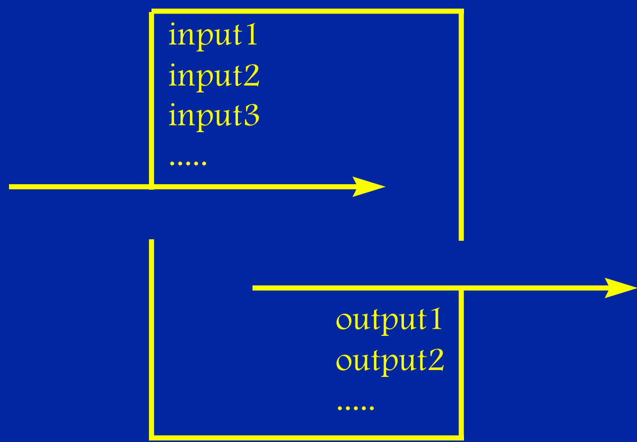
presented by

Chris Buckalew

Engines

- "Function boxes" : take inputs and produce outputs
- Inputs and outputs can include time or geometry
- Encapsulate time and geometry changes in the scene graph
- Engines may be cascaded

Under the Hood



Engines

PORTABLE GRAPHICS

Some Engine Classes

- Time inputs:
  - SoElapsedTime, SoOneShot, SoTimeCounter
- Triggered inputs:
  - SoCounter, SoOnOff, SoTriggerAny, SoGate
- SoCompose and SoDecompose
- SoInterpolate
- SoCalculator

Field Connections

- connectFrom(SoField *field);
- connectFrom(SoEngineOutput *engineOut);
- Example:

```
SoSphere *ball = new SoSphere;
SoCube *box = new SoCube;
box->width.connectFrom(&ball->radius);
root->addChild(ball);
root->addChild(box);

SoElapsedTime *counter = new SoElapsedTime;
ball->radius.connectFrom(&counter->timeOut);
```

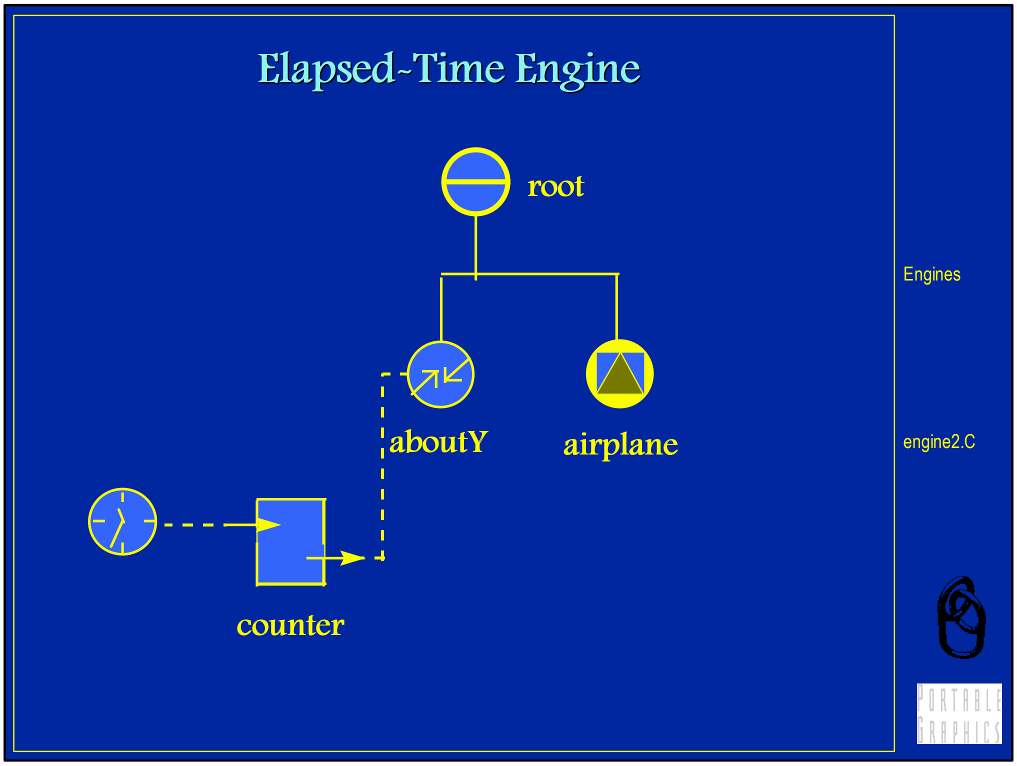
Making Field Connections

- Connection cycles
- Multiple connections
- Field type conversions
- disconnect( )
- isConnected( )

PORTABLE GRAPHICS

Time Input Engines

- SoElapsedTime - counts from start in float seconds
  - inputs: timeIn, spped, on, pause, reset
  - output: timeOut
- SoOneShot - runs for a set time, then stops
  - inputs: timeIn, duration, trigger, flags, disable
  - outputs: timeOut, isActive, ramp
- SoTimeCounter - cycles through a count in integer seconds
  - inputs: timeIn, min, max, step, on, frequency, duty, reset, syncIn
  - outputs: output, syncOut

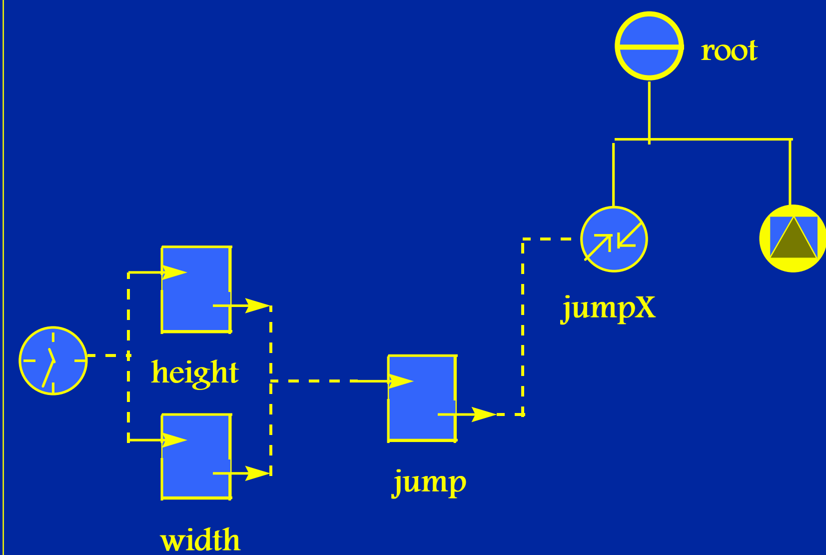

Elapsed-Time Engine
root
aboutY
airplane
counter
Engines
engine2.C
PORTABLE GRAPHICS

Elapsed-Time Example

```
SoRotationXYZ  *aboutY = new SoRotationXYZ;
aboutY->axis = SoRotationXYZ::Y;
 ...
root->addChild(airplane);
 ...
SoElapsedTime  *counter = new SoElapsedTime;
aboutY->angle.connectFrom(&counter>timeOut);
```


PORTABLE GRAPHICS

Time-Counter and Compose Engines



Engines

engine3.C

Time-Counter and Compose Engines Example

```
SoTranslation  *jumpX = new SoTranslation;
SoTimeCounter  *height = new SoTimeCounter;
SoTimeCounter  *width = new SoTimeCounter;
SoComposeVec3f  *jump = new SoComposeVec3f;

height->max = 14.0;   height->frequency = 1.0;
width->max = 30;      width->frequency = 0.15;

jump->x.connectFrom(&width->output);
jump->y.connectFrom(&height->output);
jumpX->translation.connectFrom(&jump->vector);
```

Calculator Engine Example

```
SoElapsedTime  *counter = new SoElapsedTime;
SoCalculator  *calcJump = new SoCalculator;
calcJump->a.connectFrom(&counter->timeOut);

calcJump->expression.set1Value(0, "ta = 0.25 * a");
calcJump->expression.set1Value
                    (1, "tb=5 * fabs(cos(a/2.0))");     //y coord
calcJump->expression.set1Value
                     (2, "oA = vec3f(ta, tb, 0)");       //vector

 jumpX->translation.connectFrom(&calcJump->oA);
```

Engines

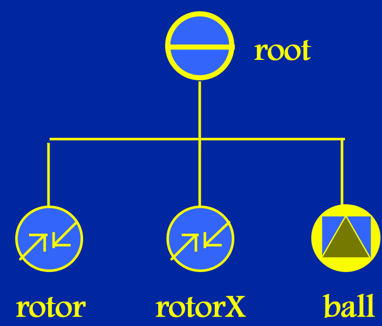
engine4.C

Gate Engine Example

```
mouseCB(void *data, SoEventCallback *eventCB) {
   SoGate *gate = (SoGate *) data;
   const SoEvent *event = eventCB->getEvent( );
   if (SO_MOUSE_PRESS_EVENT(event, ANY))
      if (gate->enable.getValue( ) == TRUE)
         gate->enable.setValue(FALSE);
      else
         gate->enable.setValue(TRUE);
}
main( ) {
....
   SoElapsedTime *counter = new SoElapsedTime;
   SoGate *gate = new SoGate(SoMFFloat::getClassTypeId( ));
   gate->input->connectFrom(&counter->timeOut);
   aboutY->angle.connectFrom(gate->output);
....
```


engine5.C

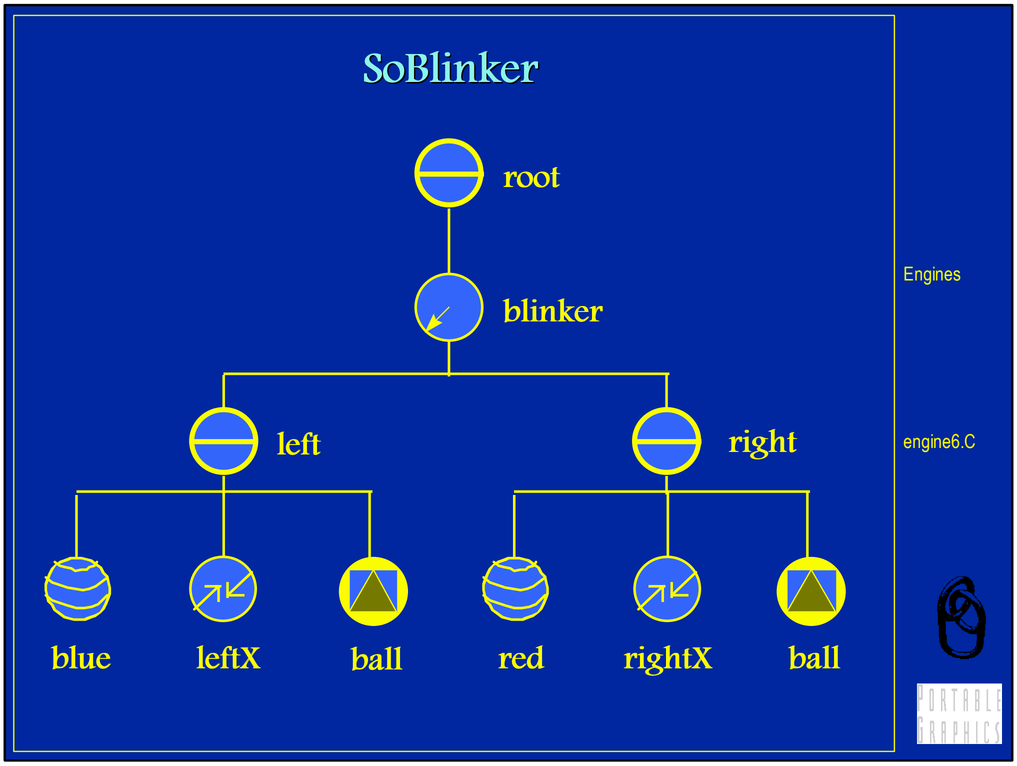
SoRotor




```
rotor->rotation.setValue(SbVec3f(0, 0, 1), 0);
rotor->speed = 0.2;
```

SoPendulum Example

```
SoPendulum  *pendulum = new SoPendulum;

pendulum->leftExtent.setValue(SbVec3f(0, 0, 1),
                              M_PI*1.5 - M_PI/4);
pendulum->rightExtent.setValue(SbVec3f(0, 0, 1),
                              M_PI*1.5 + M_PI/4);

pendulum->speed = 0.2;
```


SoBlinker
root
blinker
left
right
blue
leftX
ball
red
rightX
ball
Engines
engine6.C
PORTABLE GRAPHICS

Nodekits

presented by

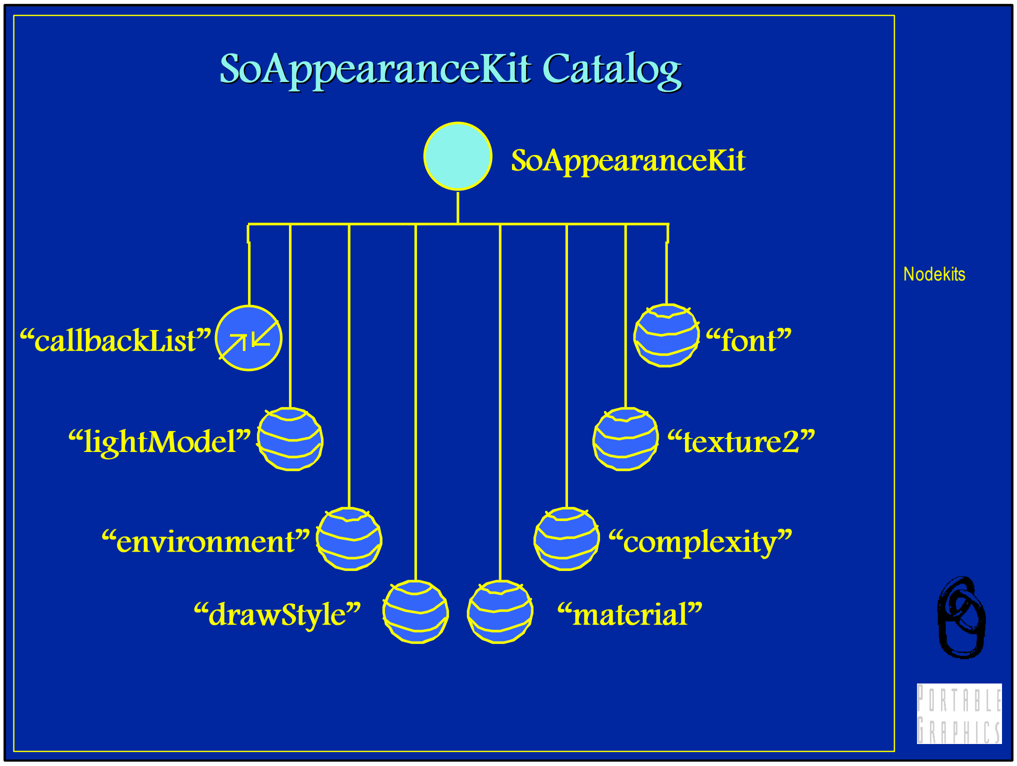
Lew Hitchner

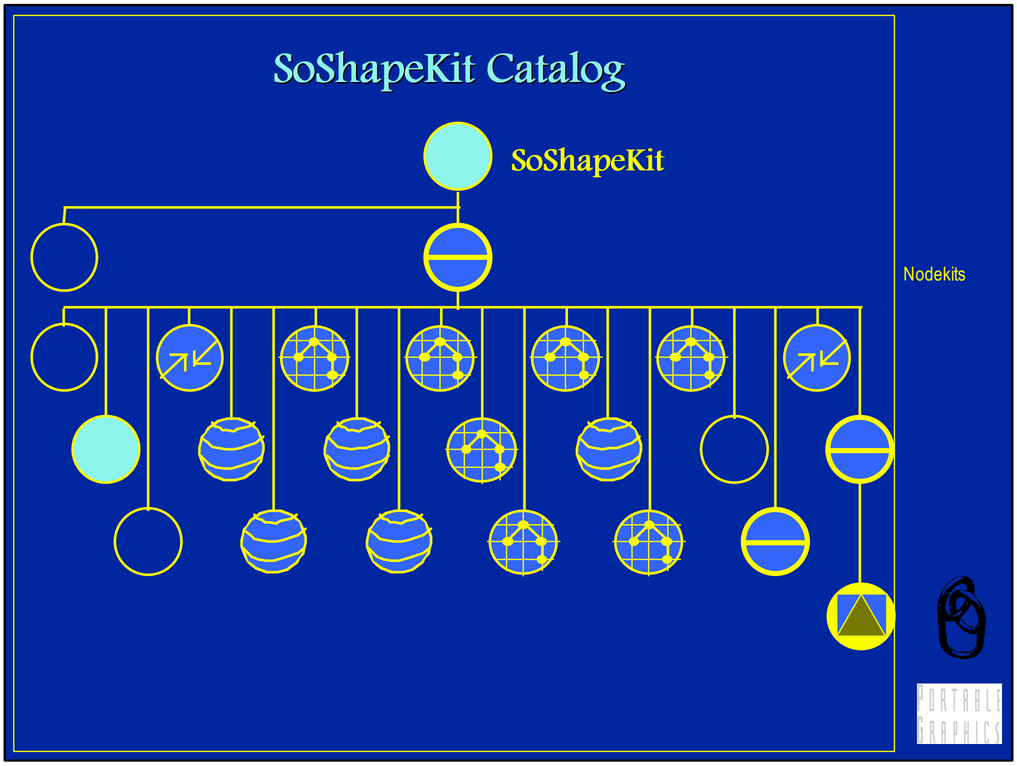
Node Kits

- Modular organization of nodes into subgraphs
- Efficient collections of nodes
- Uniform structure
- Code to generate scene graph is shorter and easier to understand

Node Kit Classes

- SoAppearanceKit
- SoCameraKit
- SoInteractionKit
  - SoDragger
- SoSeparatorKit
  - SoShapeKit
  - SoWrapperKit
- SoLightKit
- SoSceneKit


SoAppearanceKit Catalog
SoAppearanceKit
"callbackList"
"lightModel"
"environment"
"drawStyle"
"material"
"complexity"
"texture2"
"font"
Nodekits
PORTABLE GRAPHICS


SoShapeKit Catalog
SoShapeKit
Nodekits
PORTABLE GRAPHICS

Adding Parts


```
SoShapeKit  *boxKit = new SoShapeKit;

boxKit->set(“material {diffuseColor 0.5 0.5 1.0}”);

boxKit->set(“material”, “diffuseColor 0.5 0.5 1.0”);

boxKit->set(“drawStyle {style LINES}”
            “transform {scaleFactor 2.0 1.0 1.0}”);
```

Node Kit Example

```
// creating a scene graph as before
SoSphere  *globe = new SoSphere;
SoMaterial  *globeMat = new SoMaterial;
globeMat->diffuseColor.setValue(0.5, 0.5, 1.0);
SoTransform  *globeX = new SoTransform;
gloveX->scaleFactor.setValue(2.0, 1.0, 1.0);
root->addChild(globeMat);
root->addChild(globeX);
root->addChild(globe);

// using a Node Kit
SoShapeKit  *globeKit = new SoShapeKit;
globeKit->setPart(“shape”, new SoSphere);
globeKit->set(“material {diffuceColor 0.5 0.5 1.0}”
                     “transform {scaleFactor 2.0 1.0 1.0}”);
root->addChild(globeKit);
```

Nodekits

nodekit1.C

PORTABLE GRAPHICS

getPart( )

- Returns a pointer to named part in the node kit
- TRUE creates part if not there; FALSE does not
- Examples:

```
box = (SoShapeKit  *) boxKit->getPart("shape");

SoTransform  *dragX;
dragX = (SoTransform)(boxKit->
                      getPart("transform", TRUE));
```

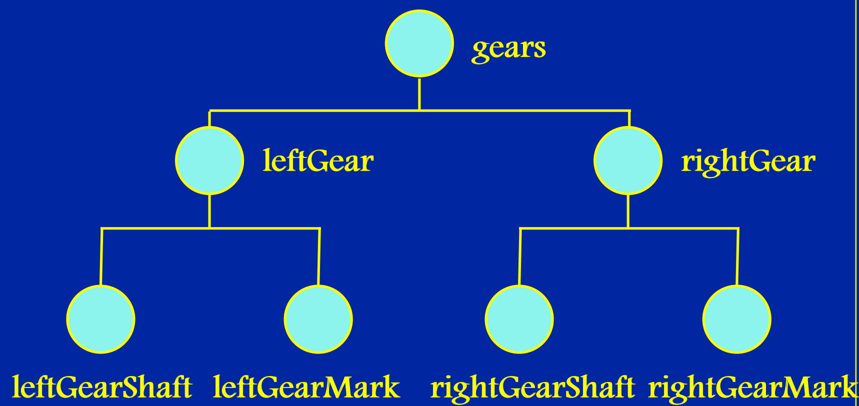
Nodekits

setPart( )

- Inserts node into node kit
- NULL pointer deletes node
- Examples:

```
boxKit->setPart("shape", new SoSphere);

SoMaterial  *newMat = new Material;
boxKit->setPart("material", newMat);

boxKit->setPart("transform", NULL);
```

Motion Hierarchy Scene Graph



Nodekits

nodekit2.C

Motion Hierarchy Example

```
SoShapeKit  *gears = new SoShapeKit;
gears->setPart(“shape”, NULL);
gears->set(“transform” {rotation 1 0 0 -0.7854}”);
root->addChild(gears);

// build left gear...
SoShapeKit  *leftGear = new SoShapeKit;
leftGear->setPart(“shape”, new SoCylinder);
SoCylinder  *cyl = (SoCylinder  *)
                    leftGear->getPart(“shape”,TRUE);
cyl->radius = 2.0;
cyl->height = 0.3;
gears->setPart(“childList[0]”, leftGear);
```

Nodekits

nodekit2.C

Draggers and Manipulators

presented by

Lew Hitchner

Draggers

- Nodes in scene graph with special geometry and user interface
- Connect dragger fields to node fields or engines
- Callback functions can be invoked when dragger interaction starts or stops, when the mouse moves, or when dragger fields change
- Build complex draggers from simple ones
- Create new draggers for different geometries
- Draggers may be connected to anything, not just geometry fields

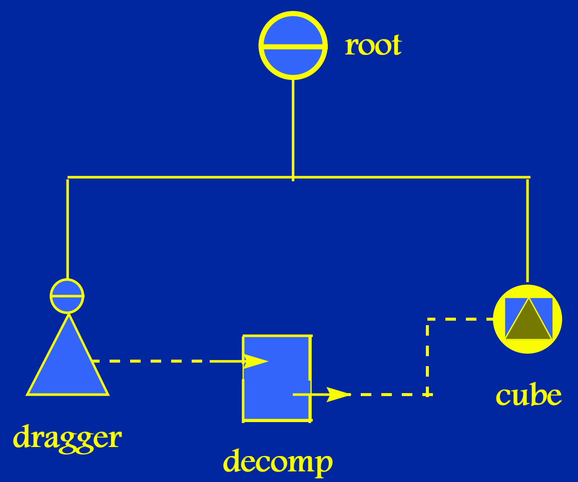
PORTABLE GRAPHICS

Dragger Classes

- Translations:
  - SoTranslate1Dragger, SoTranslate2Dragger
- Scales:
  - SoScale1Dragger, SoScale2Dragger, SoScale2UniformDragger, SoScaleUniformDragger
- Rotates:
  - SoRotateCylindricalDragger, SoRotateDiskDragger, SoRotateSphericalDragger
- Combos:
  - SoTrackballDragger, SoJackDragger, SoHandleBoxDragger, SoTransformBoxDragger, etc
- Lights:
  - SoSpotLightDragger, SoPointLightDragger, etc

dragger1.C

Simple Dragger Scene Graph



Simple Dragger Example

```
SoTranslate1Dragger  *dragger = new SoTranslate1Dragger;
root->addChild(dragger);

SoCube  *cube = new SoCube;
root->addChild(cube);

//hook dragger to engine
SoDecomposeVec3f  *decomp = new SoDecomposeVec3f;
decomp->vector.connectFrom(&dragger->translation);

// ... and hook engine to cube
cube->width.connectFrom(&decomp->x);
```


dragger2.C

Dragger Callbacks

- Start callbacks:
  - addStartCallback( ), removeStartCallback( )
- Motion callbacks:
  - addMotionCallback( ), removeMotionCallback( )
- Value-changed callbacks:
  - addValueChangedCallback( ), removeValueChangedCallback( )
- Finish callbacks:
  - addFinishCallback( ), removeFinishCallback( )

Dragger Callback Example

```
void dragStartCB(void  *mat, SoDragger  *) {
   ((SoMaterial  *)mat)->
               diffuseColor.setValue(1.0, 0.5, 0.5);}

void dragEndCB(void  *mat, SoDragger  *) {
   ((SoMaterial  *)mat)->
               diffuseColor.setValue(0.5, 1.0, 1.0);}
main( ) {
   SoMaterial  *boxColor = new SoMaterial;
   boxColor->diffuseColor.setValue(0.5, 1.0, 1.0);
   SoTranslate1Dragger  *drag = new SoTranslate1Dragger;
   drag->addStartCallback(dragStartCB,  boxColor);
   drag->addFinishCallback(dragEndCB,  boxColor);
```


dragger3.C

Multiple Dragger Example

```
SoTransform  *xTrans = new SoTransform;
xTrans->translation.setValue(0.0, -2.0, 4.0);
SoTranslate1Dragger  *xDragger = new SoTranslate1Dragger;
xSep->addChild(xTrans);        xSep->addChild(xDragger);
root->addChild(xSep);
...
SoTransform  *scale = new SoTransform;        root->addChild(scale);
SoSphere  *ball = new SoSphere;               root->addChild(ball);

SoCalculator  *calc = new SoCalculator;
calc->A.connectFrom(&xDragger->translation);
calc->B.connectFrom(&yDragger->translation);
calc->C.connectFrom(&zDragger->translation);
calc->expression = “oA = vec3f( A[0],  B[0],  C[0])”;
scale->scaleFactor.connectFrom(&calc->oA);
```


dragger4.C

Manipulators

- Manipulators are instances of nodes with interactively editable geometry

dragger5.C

Swapping Nodes and Manipulators

- replaceNode( ) method replaces a node with its editable version
- replaceManip( ) restores the node with its new values
- Example:

```
trackBall = new SoTrackballManip;
pathX = createPathtoTransform(pickPath);
track->replaceNode(pathX);
...
track->replaceManip(pathX, new SoTransform);
```

Types of Manipulators

- SoTransformManip replaces transformations
- SoPointLightManip, SoDirectionalLightManip, and SoSpotLightManip replace lights
- Manipulators may be customized by replacing geometry, but functionality may not be changed

Cool Nodes and other topics

presented by

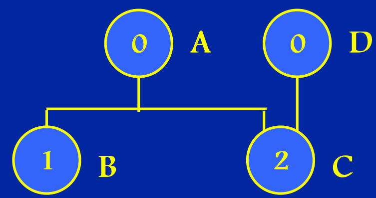
Chris Buckalew

PORTABLE GRAPHICS

Node References

- Create nodes with the ***new*** operator, rarely on the stack
- A node’s reference count is incremented by:
  - adding the node to a group
  - including the node in a path
  - manual reference with ***ref( )***
- A node’s reference count is decremented by:
  - removing the node from a group
  - deleting a path containing the node
  - manual dereference with ***unref( )***
- Inventor automatically deletes a node when its reference count is decremented to zero

Managing the Scene Database



Problem: how to remove B from A and add to D?

Solutions: add B to D first, or manually increment B’s count

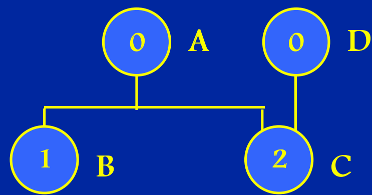
```
D->addChild(B);
A->removeChild(B);
```

```
B->ref( );
A->removeChild(B);
D->addChild(B);
```

Scene Database

PORTABLE GRAPHICS

Managing the Scene Database (cont)

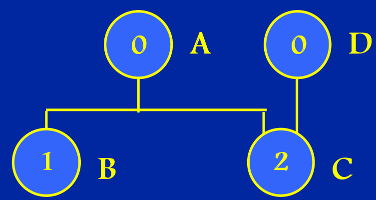


Problem: how to delete A?

Solution: manually decrement A’s reference count. This also decrements B and C by one

A->unref( );

Managing the Scene Database (cont)



Problem: Actions such as rendering create then delete paths. This would increment then decrement A's reference count back to 0, deleting A.

Solution: manually increment A's reference count. This must ordinarily be done to root nodes.

Managing the Scene Database (cont)

- Never allocate nodes, paths, or engines in arrays - causes problems when Inventor tries to delete one element from the array
- Never declare nodes, paths, or engines on the stack - may cause problems becasue nodes may go out of scope before Inventor removes them from the scene database.
- Common problem is trying to access a node that has been automatically deleted by Inventor.

Common Problem Example

```
SoSphere  *buildSphere( ) {
   SoSphere  *ball = new SoSphere;
         // ref count now 0

   ba = new SoGetBoundingBoxAction;
   ba->apply(ball);
         // ref count goes to 1 then back to 0
   box = ba->getBoundingBox;

         // since ref count decremented to 0, ball is deleted
   return ball;
         // here comes a core dump
}
```


PORTABLE
GRAPHICS

Virgin Nodes

Sometimes we want to use a node but keep its reference count at zero:

```
SoSphere  *buildSphere( ) {
   SoSphere ball = new SoSphere;
   ball->ref( );

   ba = new SoGetBoundingBoxAction;
   ba->apply(ball);
   box = ba->getBoundingBox;

   ball->unrefNoDelete( );
   return ball;
```

Subclasses of SoGroup

- SoSwitch and SoBlinker
  - traverses only one child
- SoLOD
  - different object resolutions at different distances
- SoArray and SoMultipleCopy
  - traverse multiple copies of children
- SoPathSwitch
  - traverses children if current path matches path field
- SoTransformSeparator
  - saves and restores only transform state
- SoAnnotation
  - children traversed after rest of scene graph

SoSwitch and SoBlinker

- SoSwitch visits only one of its children, determined by the whichChild field
- Default: whichChild = SO_SWITCH_NONE
- Example:

```
SoSwitch  *switch = new SoSwitch;
switch->addChild(A);
switch->addChild(B);
switch->insertChild(C, 1);
switch->whichChild = 2;
```

- SoBlinker includes an engine which automatically cycles through children

SoLOD

- Specify same object with different levels of detail
- SoLOD is a subclassed group node; only one child is traversed based on distance to camera
- range: array of floats determine changeover
  points
- center: point in object space with which distance
  to camera is computed

Cool Nodes

nodes1.C

PORTABLE GRAPHICS

SoDrawStyle

- style:
  - FILLED, LINES, POINTS, INVISIBLE
- pointSize:
  - radius of points; units are printer's points
  - default is 0.0; uses fastest value for rendering
- lineWidth:
  - width in points, default is 0.0
- linePattern:
  - 0x0 to 0xffff for invisible to solid
- Points and lines are best rendered with BASE_COLOR lighting

SoLightModel

- model:

  BASE_COLOR - ignores light sources and uses only diffuseColor and transparency values for shading

  PHONG - uses all lights and surface normals to compute shading

SoEnvironment

- ambientIntensity
- ambientColor
- attenuation: vector of squared, linear, and constant attenuation with distance from lights
- fogType:
  - NONE
  - HAZE: opacity linear with distance
  - FOG: opacity exponential with distance
  - SMOKE: opacity exponential-squared with distance
- fogColor
- fogVisibility: distance at which objects are totally obscured

Cool Nodes

nodes4.C

PORTABLE GRAPHICS

SoComplexity

- Governs amount of tesselation for spheres, cylinders, NURBS, etc.
- Fields:
  - value: 0.0 is minimum tesselation and 1.0 is maximum
  - type:
    - OBJECT_SPACE
    - SCREEN_SPACE
    - BOUNDING_BOX
  - textureQuality: filtering level
- Example shows SCREEN_SPACE and OBJECT_SPACE complexities

nodes5.C

ShapeHints Node

- vertexOrdering
  - UNKNOWN_ORDERING
  - CLOCKWISE
  - COUNTERCLOCKWISE
- shapeType
  - UNKNOWN_SHAPE_TYPE
  - SOLID
- faceType
  - UNKNOWN_FACE_TYPE
  - CONVEX
- creaseAngle: adjacent facets share normal if angle between normals less than this field

PORTABLE GRAPHICS

SoUnits

- Automatically scales objects with different units so that they all display at the correct size
- unit:

METERS	CENTIMETERS
MILLIMETERS	MICROMETERS
MICRONS	NANOMETERS
ANGSTROMS	KILOMETERS
FEET	INCHES
POINTS	YARDS
MILES	NAUTICAL_MILES

Inventor File Format

- Inventor's file format is used for reading and storing scene graphs, paths, or nodes to and from ASCII files
- Users can edit files rather than edit and recompile programs
- Complex scene geometry may be read in from files modularly
- File format is used for cutting and pasting between windows or processes
- File format is also used to specify node kit parts

Reading a File into the Database

```
SoNode*  readIvFile(const  char  *filename) {
   SoInput  sceneInput;
   SoDB::init( );
   if (! sceneInput.openFile(filename))
      cout<<"problem opening file"<<filename;
   SoSeparator *node = SoDB::readAll(&sceneInput);
   if (! node) {
      cout<<"problem reading file"<<filename;
   sceneInput.closeFile( );
   return node;
}
```

File Format Example

```
Separator {
   PerspectiveCamera {position 0 0 -3.4496}
   DirectionalLight{ }
   Transform {
      translation 3.89 -7.5  6.0
      scaleFactor 1.0  1.0  2.5 }
   Separator {
      Material {diffuseColor  1.0  0.5  1.0}
      Sphere { }
   }
}
```


PORTABLE GRAPHICS

Different Formats

- Formats for writing:
  - Engines
  - Field connections
  - Global fields
  - Shared instances of nodes
  - Paths
  - Node kits
- Can also read from a string

Event Handling

presented by

John Readey

PORTABLE GRAPHICS

Input Processing & Events

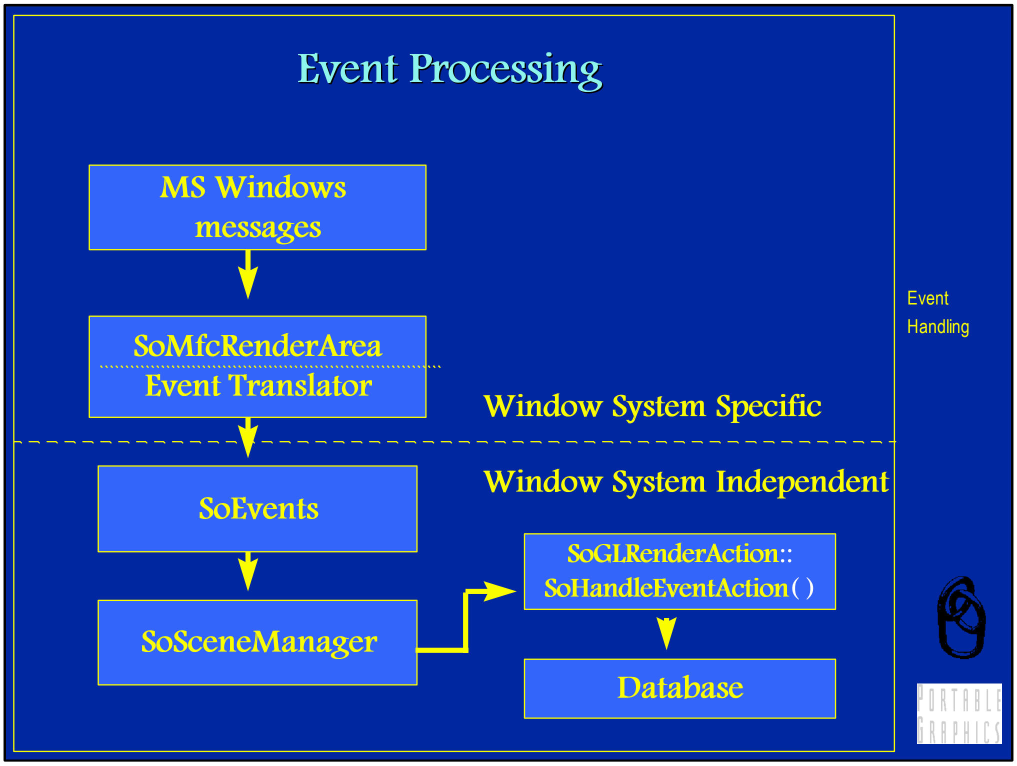
- Events are generated by the keyboard and mouse (or other input devices)

- Many Inventor classes respond to events (e.g. Manipulators)

- The developer can overide the default behavior by
  - Subclassing an Inventor class and modifying the behavior
  - Intercepting the event before it is processed by Inventor

Inventor Events

- Window specific events (XEvents in UNIX, messages in Windows NT/95) are translated by the component library into Inventor specific SoEvents.
- Each SoEvent instance contains information on:
  - Type type of event (keyboard, mouse button, mouse move, etc)
  - The time the event occured
  - The cursor position at the time of the event
  - The state of the modifier keys (control, shift, alt) when the event ocurred


Event Processing
MS Windows messages
SoMfcRenderArea
Event Translator
Window System Specific
Window System Independent
SoEvents
SoSceneManager
SoGLRenderAction::
SoHandleEventAction( )
Database
Event Handling
PORTABLE GRAPHICS

Picking using the SoSelection Node

- The SoSelection node is a group class that is typically inserted near the top of the scene graph. It handles any event that its children don't handle.
- It maintains a selection list of picked objects according to a policy set in the policy field:
  - SINGLE: one object at a time, mouse pick on nothing clears selection
  - TOGGLE: multiple objects, left mouse pick toggles selection status
  - SHIFT: when shift key is down, policy is TOGGLE; when shift key is up, policy is SINGLE

Selection Example

```
main( )
SoSelection  *sel = new SoSelection;
sel->policy = SoSelection::SHIFT;
sel->addSelectionCallback(selectCB, highMat);
...
}
void selectCB(void  *data, SoPath  *selectionPath) {
   SoMaterial  *highMat = (SoMaterial  *) data;
   if (selectionPath->getTail( )->isOfType(..sphere..)
      highMat->transparency = 0.4;
}
```

Event Handling

events1.C

Handling Events with Callback Nodes

- The SoEventCallback node can be inserted into a scenegraph to provide application specific behavior. The developer can provide a callback function that will be invoked whenever the node receives an event of the proper type.
- Example:

```
// An event callback node so we can receive key press
// events
SoEventCallback *myEventCB = new SoEventCallback;
myEventCB->addEventCallback(
      SoKeyboardEvent::getClassTypeId(),
      myKeyPressCB, selectionRoot);
selectionRoot->addChild(myEventCB);
```

Bypassing Inventor Event Handling

- The application can intercept events (in the native window system dependent format) before Inventor receives them.

- Events can also be processed by the application and then passed on to the Inventor event handling mechanism.

SoMFC Event Handling Example

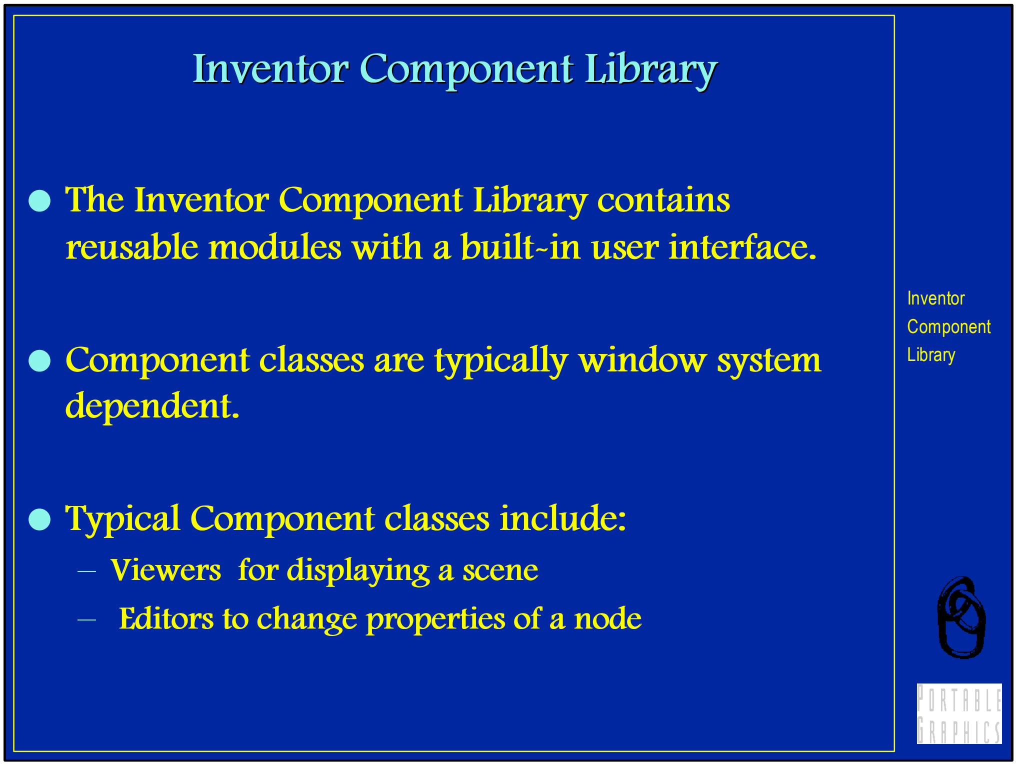
```
void
CDropView::OnMouseMove(UINT nFlags, CPoint point)
{
    movement[0] = locator[0];
    movement[1] = locator[1];
    locator[0] = windowSize[0] - point.x;
    locator[1] = windowSize[1] - point.y;
    if (mode == TRANS_MODE)
        translateCamera( );
    else if (mode == ROT_MODE)
        rotateCamera( );

    SoMfcView::OnMouseMove(nFlags, point);
}
```

Interfacing to the Windowing System

presented by

John Readey

PORTABLE GRAPHICS


Inventor Component Library
The Inventor Component Library contains reusable modules with a built-in user interface.
Component classes are typically window system dependent.
Typical Component classes include:
– Viewers for displaying a scene
– Editors to change properties of a node
Inventor Component Library
PORTABLE GRAPHICS

Component Classes for Windows NT/95

- PGI supplies two different component libraries with Open Inventor for Windows NT
  - WinSoXt
  - SoMFC
- WinSoXt includes classes compatible with the SoXt classes on UNIX versions of Inventor.
- WinSoXt classes are:
  - Familiar (if you have experience with SoXt)
  - Easy to use (your entire program can be just a dozen lines)
- The dark side:
  - You don't have access to native Windows features.

Xt Components

- Editors:
  - SoXtMaterialEditor
  - SoXtMaterialList
  - SoXtLightSliderSet
  - SoXtMaterialSliderSet
  - SoXtTransformSliderSet
- Viewers:
  - SoXtFullViewer
  - SoXtFlyViewer
  - SoXtWalkViewer
  - SoXtExaminerViewer
  - SoXtPlaneViewer

MaterialEditor Example

- Pass values back with a callback, OR
- Attach the editor to a node directly

```
SoXtMaterialEditor  *headEdit = new SoXtMater…;
SoMaterial  *headM = new SoMaterial;
headEdit->attach(headM);

renderArea->show( );
SoXt::show(window);
headEdit->show( );
SoXt::mainLoop( );
```

SoMFC Components

- MFC is a widely popular C++ class library developed by Microsoft and used for Windows-based application development.
- SoMFC is an MFC Extension library that enables MFC based applications to incorporate Inventor.
- It includes more than 30 classes that provide:
  - Viewers
  - Editors
  - Printing Support
  - OLE Integration

The Document/View Architecure

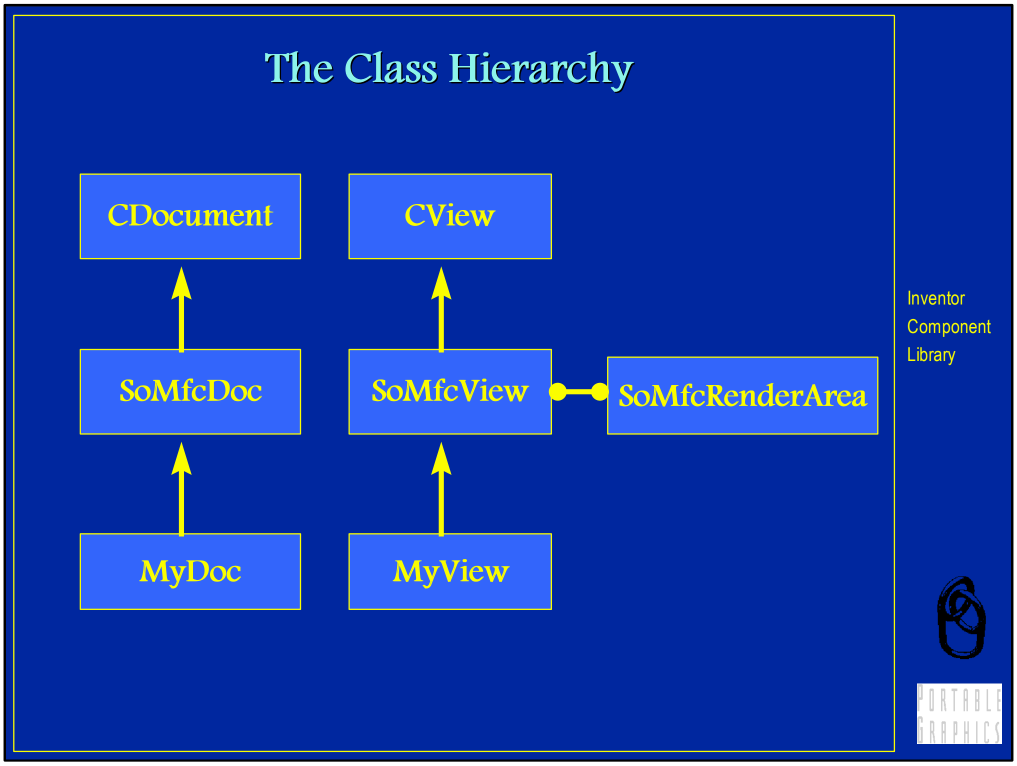
- The document and view classes are fundamental to MFC.
- Document classes are used to encapsulate the ***data*** an application deals with.
- View classes encapsulate how the data is presented to the user.
- Applications can be
  – SDI (just one document and one view)
  – MDI (multiple documents and views)

Inventor
Component
Library

PORTABLE
GRAPHICS

Integration of MFC and Inventor

- MFC applications typically consist of application specific view and document classes derived from the MFC classes CView and CDocument (or their analogues).
- To create an Inventor SoMFC application, the user creates classes derived from the SoMFC classes, SoMfcView and SoMfcDocument instead of CView and CDocument.
- The viewer instance (SoMfcRenderArea, SoMfcViewer, SoMfcExaminerViewer, etc) is contained in the SoMfcView object.


The Class Hierarchy
CDocument
CView
SoMfcDoc
SoMfcView
SoMfcRenderArea
MyDoc
MyView
Inventor
Component
Library
PORTABLE
GRAPHICS

SoMfc Editor Classes

- Editor classes include
  - SoMfcColorEditor
  - SoMfcMaterialEditor
  - SoMfcHeadlightEditor
  - SoMfcMaterialPalette
  - SoMfcTextureMapEditor
- Most of these classes have an interface and functionality similar to their Xt counterparts.

SoMfc Viewer Classes

- SoMfc Viewer classes are always contained within SoMfcView.
- They include
  - SoMfcRenderArea
  - SoMfcViewer
  - SoMfcExaminerViewer
  - SoMfcFlyViewer
  - SoMfcPlaneViewer
  - SoMfcWalkViewer

Using the Windows Clipboard

- Inventor scene objects can be cut or pasted into the clipboard.
- When pasted into a non-Inventor application an ascii-based file description of the nodes will be displayed.
- When pasted into an Inventor application, the nodes can be added to the current scene graph.

Inventor and OLE

- OLE is an architecture developed by Microsoft that allows different applications to inter-operate.
- The OLE vision is to focus on documents, rather than applications.
- A common application of OLE is Object Linking and Embedding. This allows an instance of an OLE Server app to be placed into any OLE Client Application.
- Inventor based OLE Server applications can be embedded or linked into OLE Client applications such as Microsoft Word or Excel.

PORTABLE GRAPHICS

Optimizing Open Inventor

presented by

John Readey

PORTABLE GRAPHICS

Optimizing Performance

- These are some simple guidelines to performance tuning
- Keep in mind that performance characteristics will vary with platform and graphics adapter

Turn Culling On if Possible

- For parts of the scenegraph that consist of all closed surfaces, turn backface culling on with the ShapeHints node.
- For scenegraphs that contain shapes spread across a large volume (e.g. a model of the solar system), turn viewport culling on.

Use Shared Instancing

- If the same object is used repeatedly in your scene graph, create only one instance of it
- This is especially important for SoTexture2 nodes

Use the new Vertex Property Node

- SoVertexProperty was a new node introduced with Inventor 2.1
- The SoVertexProperty node is an efficient way to specify attributes for vertex-based shape nodes
- Properties that can be set include: coordinates, normals, colors, transparency, material and normal binding
- Specify all fields for maximum performance

PORTABLE GRAPHICS

Vertex Property Node Example

```
SoVertexProperty *earVP = new SoVertexProperty;
// define material binding
earVP->normalBinding =
   SoNormalBinding::PER_FACE;
// define the coordinates
earVP->vertex.setValues(0, 9, earVerts);
// define the colors
earVP->orderedRGBA.setValues(0, 8, earColors);
...
earFaceSet-> vertexProperty.setValue(earVP);
```

Optimize your .iv files!

- ivFix is a new utility provided with Inventor 2.1.1 that will re-organize your .iv file for maximum performance
- ivAddVP can be used to transform .iv files to use the SoVertexProperty Node
- Run ivPerf to analyze performance

The Future of Open Inventor

presented by

Silicon Graphics, Inc

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.sd.tgs.com
  - VRML home page: vrml.wired.com

Other Topics

PORTABLE GRAPHICS