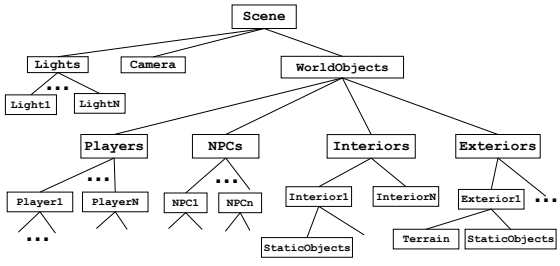
CSc 165
Computer Game Architecture

6 - Scenegraphs

Overview

- The Scenegraph Concept
- Common Scenegraph APIs
- Scenegraph Node Hierarchy
- Scenegraph Traversal
- Node Controllers
- Render Queues

Hierarchical Scenegraphs

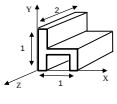


Common Scenegraph APIs

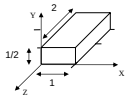
- OpenSceneGraph
  `www.openscenegraph.org`
- Java3D (Sun/Java Community)
  `https://java3d.dev.java.net`
- X3D (Web3D Consortium)
  `www.web3d.org/x3d`
- OpenSG
  `http://www.opensg.org`
- OpenInventor (SGI)
  `http://oss.sgi.com/projects/inventor`
- Xith3D
  `https://xith3d.dev.java.net`

Building a Scenegraph (1)

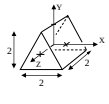
Primitive objects ("models") defined in "local space":



"Bench"

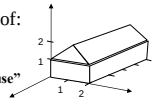


"Wall"



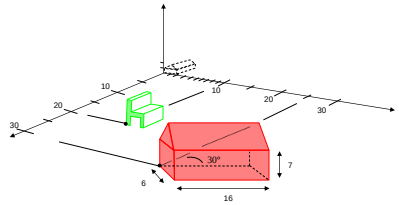
"Roof"

Hierarchical object – House = Wall + Roof:



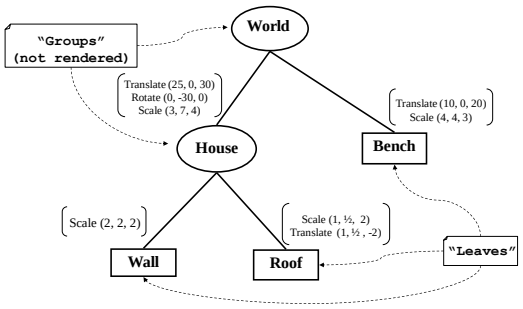
"House"

Building a Scenegraph (2)

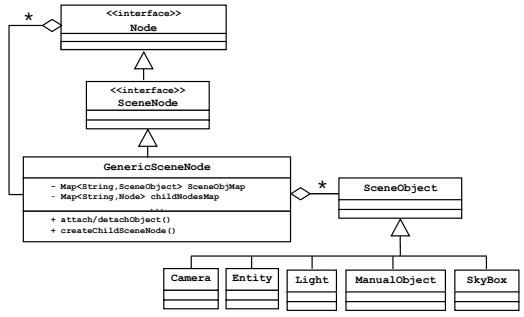



```
World = House ( Transform = ([Translate(25,0,30)] x [Rotate(0,-30,0)]
                x [Scale(3,7,4)]),   Color = Red )
        +
        Bench ( Transform = ([Translate(10,0,20)] x [Scale(4,4,3)]),
                Color = Green)
```

Building a Scenegraph (3)



Scenegraph Nodes



(Naïve) Scenegraph Traversal

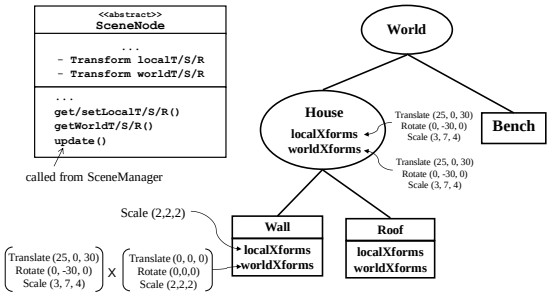
Renderer :

```
public void displayScenegraph()
{   ...
    save current xform matrix
    sceneGraphRoot.draw()
    restore xform matrix
    ...
}
```

SceneNode.draw() :

```
public void draw()
{   save r.xform
    concatenate localXforms onto r.xform
    if (this.visible()) render(this)
    for (each child of this group)
    {   child.draw()
    }
    restore r.xform
}
```


Improved Scenegraph Structure



Example Game Application

```
public class MyGame extends VariableFrameRateGame
{   protected void setupScene()
    {   //create some scene objects (children of the scenegraph root)
        Entity dolphinE = sm.createEntity("myDolphin", "dolphinHighPoly.obj");
        dolphinE.setPrimitive(Primitive.TRIANGLES);
        SceneNode dolphinN =
            sm.getRootSceneNode().createChildSceneNode("myDolphinNode");
        dolphinN.attachObject(dolphinE);

        Entity earthE = sm.createEntity("myEarth", "earth.obj");
        earthE.setPrimitive(Primitive.TRIANGLES);
        SceneNode earthN = dolphinN.createChildSceneNode("myEarthNode");
        earthN.attachObject(earthE);
        ...
    }

    ...continued...
```


Updating Nodes' Positions

(done in GenericSceneNode)

```
public void update(boolean updateChildren, boolean parentHasChanged)
{   boolean updateRequired = parentHasChanged || parentOutOfSync;
    if (updateRequired)
        updateFromParent();
    if (updateChildren)
        for (Node n : childNodesMap.values())
            n.update(updateChildren, updateRequired);
}
public void updateFromParent()
{   updateWorldPosition();
    updateWorldRotation();
    updateWorldScale();
    parentOutOfSync = false;
}
private void updateWorldPosition()
{   Vector3 worldPos =
                    parentNode.getWorldPosition().
                      add(parentNode.getWorldRotation().mult(scaledPos));
    worldTransform.setPosition(worldPos);
}
```

Specifying Node Behavior

Certain operations on nodes occur often :

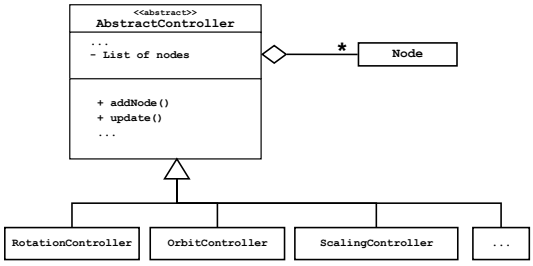
- Spatial transforms (rotate, scale, translate)
- Lifetime expiration
- Change in appearance (transparency, etc.)
- … many others …

Two approaches

- Require game application to support such changes
- Provide game engine classes to manage changes

Controllers

Controllers are attached to SceneNodes



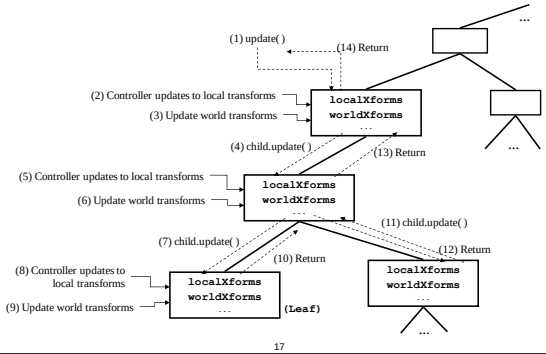
Example: StretchController

```
public class StretchController extends AbstractController
{   private float scaleRate = .003f;    // growth per second
    private float cycleTime = 2000.0f;  // default cycle time
    private float totalTime = 0.0f;
    private float direction = 1.0f;

    @Override
    protected void updateImpl(float elapsedTimeMillis)
    {   totalTime += elapsedTimeMillis;
        float scaleAmt = 1.0f + direction * scaleRate;

        if (totalTime > cycleTime)
        {   direction = -direction;
            totalTime = 0.0f;
        }

        for (Node n : super.controlledNodesList)
        {   Vector3 curScale = n.getLocalScale();
            curScale =
                Vector3f.createFrom(curScale.x()*scaleAmt, curScale.y(), curScale.z());
            n.setLocalScale(curScale);
        }
    }
}
```


Example Game (revisited)

```
public class MyGame extends VariableFrameRateGame
{  protected void setupScene()
   {  . . .
      StretchController sc = new StretchController();
      sc.addNode(dolphinN);
      sm.addController(sc);
      . . .
```


Hierarchical Update Sequence



Rendering the SceneGraph

- **`render()`** == 'draw the scene'
- Standard approach: recursive tree-walk calling **`draw()`** at each **`SceneNode`**
- Problem: Scenegraph traversal order doesn't account for differences in nodes:
  - ***Opaque*** nodes should be rendered front-to-back for speed
  - ***Transparent*** nodes must be rendered after opaque ones, and in back-to-front order
  - ***Orthographic*** nodes must use a different *projection*