KAIST

CS482 Lab Session

Scene Graph

2016. 10. 12

VISUAL COMPUTING Lab

KAIST

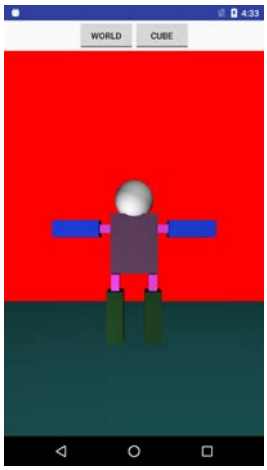
Scene Graph

VISUAL COMPUTING Lab

Scene graph

KAIST

- Building a structure to deal with objects in a smarter way



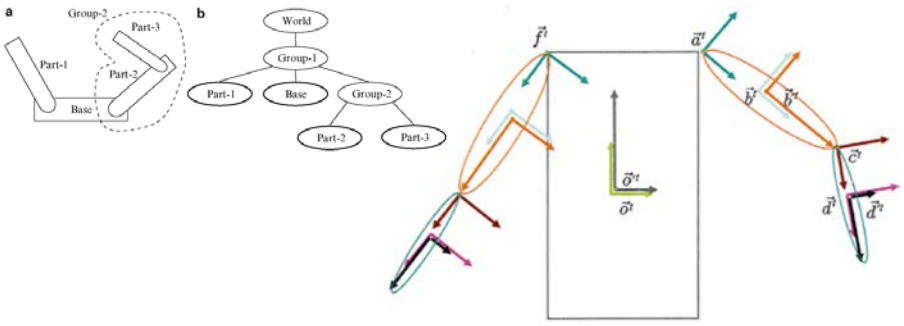
3

VISUAL COMPUTING Lab

Scene graph

KAIST

- If we move the body of the robot, the connected components should be modified automatically.
- For example, if we rotate the right arm, the right elbow and the right hand should move/rotate accordingly.



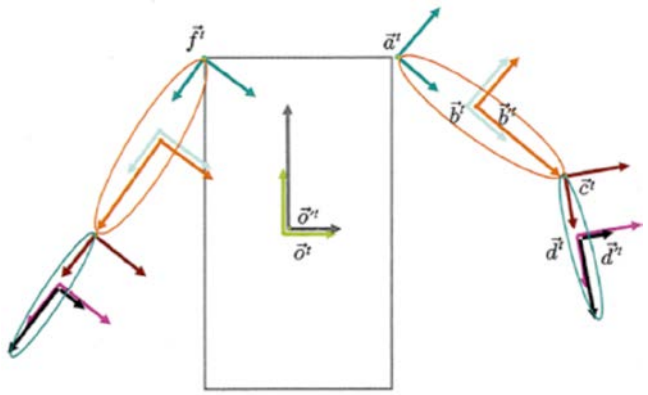
4

VISUAL COMPUTING Lab

Scene graph

KAIST

- In order to build it, we describe an object frame with the previous object frame, not the world frame.



$$\vec{\mathbf{o}}^{t} = \vec{\mathbf{w}}^{t} O$$
$$\vec{\mathbf{o}}'^{t} = \vec{\mathbf{o}}^{t} O'$$
$$\vec{\mathbf{a}}^{t} = \vec{\mathbf{o}}^{t} A$$
$$\vec{\mathbf{b}}^{t} = \vec{\mathbf{a}}^{t} B$$
$$\vec{\mathbf{b}}'^{t} = \vec{\mathbf{b}}^{t} B'$$
$$\vec{\mathbf{c}}^{t} = \vec{\mathbf{b}}^{t} C$$
$$\vec{\mathbf{d}}^{t} = \vec{\mathbf{c}}^{t} D$$
$$\vec{\mathbf{d}}'^{t} = \vec{\mathbf{d}}^{t} D'$$
$$\vec{\mathbf{f}}^{t} = \vec{\mathbf{o}}^{t} F$$

$$\vec{d}'^{t}\, c_{d'} = \vec{w}^{t} OABCD' c_{d'}$$

VISUAL COMPUTING Lab

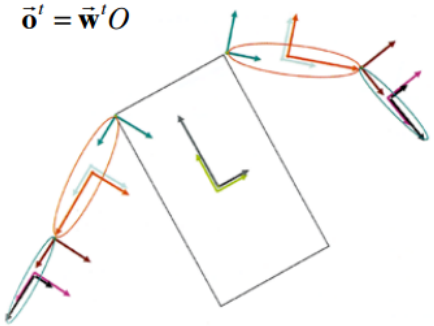
5

Scene graph

KAIST

- Other parts would be changed following the scene graph when we modify the object frame.

$$\vec{\mathbf{o}}^{t} = \vec{\mathbf{w}}^{t} O$$



$$\vec{\mathbf{o}}^{t} = \vec{\mathbf{w}}^{t} O$$
$$\vec{\mathbf{o}}'^{t} = \vec{\mathbf{o}}^{t} O'$$
$$\vec{\mathbf{a}}^{t} = \vec{\mathbf{o}}^{t} A$$
$$\vec{\mathbf{b}}^{t} = \vec{\mathbf{a}}^{t} B$$
$$\vec{\mathbf{b}}'^{t} = \vec{\mathbf{b}}^{t} B'$$
$$\vec{\mathbf{c}}^{t} = \vec{\mathbf{b}}^{t} C$$
$$\vec{\mathbf{d}}^{t} = \vec{\mathbf{c}}^{t} D$$
$$\vec{\mathbf{d}}'^{t} = \vec{\mathbf{d}}^{t} D'$$
$$\vec{\mathbf{f}}^{t} = \vec{\mathbf{o}}^{t} F$$

6

VISUAL COMPUTING Lab

KAIST

Drawing a Robot using a Scene Graph

VISUAL COMPUTING Lab

Transform nodes

KAIST

- A transform node represents a child's frame with respect to its parent frame through a rigid body transform (RBT).
- A transform node will store that RBT.
- One special transform node is the root node, which is simply the root of our entire scene graph.
- We will call it **g_world** in the codes. It corresponds to the world frame.

VISUAL COMPUTING Lab

Shape nodes

KAIST

- A shape node represents actual things that get drawn (such as a cube, a sphere, or some other piece of geometry).
- A shape node can not have any children.
- The shape to be drawn will be situated somehow with respect to its parent frame (which is a transform node).
- Since this relationship is not necessarily rigid-body, a shape node stores a Matrix4.
- Why affine transform? (not a rigid body)
  - We store the geometry object separately. Thus, we need to scale the geometry appropriately.

VISUAL COMPUTING Lab

KAIST

Scene Graph Classes

VISUAL COMPUTING Lab

Abstract / Concrete Class

KAIST

- **An abstract base class** refers to a class with unimplemented virtual functions. So they are kind of like Interfaces in Java. An abstract base type cannot be instantiated since it has unimplemented virtual methods.
- **A concrete class** has all its virtual methods implemented, and hence can be instantiated.

VISUAL COMPUTING Lab

Scene Graph Node Classes

KAIST

- Abstract Classes
  - SgNode
  - SgTransformNode
  - SgShapeNode
- Concrete Classes
  - SgRootNode
  - SgRbtNode
  - SgGeometryShapeNode

VISUAL COMPUTING Lab

SgNode

KAIST

- Abstract base class of all scene graph nodes. It defines a comparison operators (==, !=) testing whether two scene graph nodes are the same node, which might come handy at times.
- It declares a virtual function accept(...) which needs to be implemented by all derived types to support the so called Visitor pattern.

VISUAL COMPUTING Lab

SgTransformNode

KAIST

- Abstract base class of all transform nodes.
- Derives from ***SgNode***.
- Recall that transform node encodes an RBT that transforms from a parent frame to the current frame.
- Thus ***SgTransformNode*** declares a virtual function ***getRbt()*** returning this RBT.
- Since ***SgTransformNode*** can have children, it implements a bunch of function calls like ***addChild***, ***removeChild***, ***getNumChildren***, ***getChild(i)***.

VISUAL COMPUTING Lab

SgShapeNode

KAIST

- Abstract base class of all shape nodes.
- A shape nodes knows how to draw itself, and needs to store a (not-necessarily rigid body) affine transform.
- Hence it has two virtual methods ***draw*** and ***getAffineMatrix***.
- ***draw*** will take in the current ***ShaderState*** as argument, and draw the node itself.

VISUAL COMPUTING Lab

SgRootNode

KAIST

- Concrete class deriving from ***SgTransformNode***. This is a transform node corresponding to the root of the scene tree. Its ***getRbt()*** always returns an identity transform.

VISUAL COMPUTING Lab

SgRbtNode

KAIST

- Another concrete class deriving from ***SgTransformNode***. This is a transform node that wraps a ***RigTForm*** and allows full freedom of rotation/translation.

VISUAL COMPUTING Lab

SgGeometryShapeNode

KAIST

- A concrete class deriving from ***SgShapeNode***, parameterized by a user specified type Geometry.

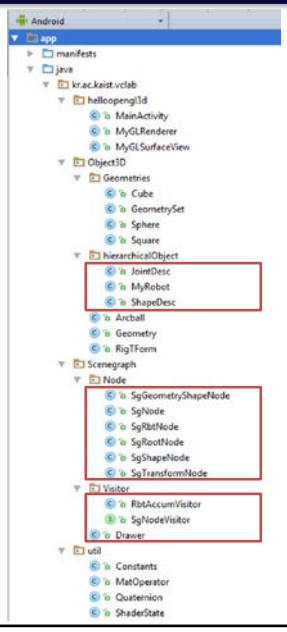
VISUAL COMPUTING Lab

KAIST

Implementation

VISUAL COMPUTING Lab

KAIST



VISUAL COMPUTING Lab

SgNode

KAIST

```java
package kr.ac.kaist.vclab.Scenegraph.Node;

import kr.ac.kaist.vclab.Scenegraph.Visitor.SgNodeVisitor;

public class SgNode extends Object{
    public boolean accept(SgNodeVisitor visitor){
        boolean result = true;
        return result;
    }

}

```


VISUAL COMPUTING Lab

SgTransformNode

KAIST

```java
package kr.ac.kaist.vclab.Scenegraph.Node;

import ...

public class SgTransformNode extends SgNode {
    private Vector<SgNode> children = new Vector<>();

    @Override
    public boolean accept(SgNodeVisitor visitor){
        if (!visitor.visit(this))
            return false;
        for (int i = 0, n = children.size(); i < n; ++i) {
            if (!children.get(i).accept(visitor))
                return false;
        }
        return visitor.postVisit(this);
    }

    public RigTForm getRbt(){
        return null;
    }
    public void setRbt(RigTForm _rbt){
    }

    public void addChild(SgNode child){
        children.add(child);
    }

    public void removeChild(SgNode child){
        children.remove(child);
    }

    public int getNumChild(){
        return children.size();
    }

    public SgNode getChild(int i){
        return children.get(i);
    }
}

```


VISUAL COMPUTING Lab

SgShapeNode

KAIST

```
1  package kr.ac.kaist.vclab.Scenegraph.Node;
2
3  import ...
5
6  public class SgShapeNode extends SgNode {
7      @Override
8      public boolean accept(SgNodeVisitor visitor){
9          if (!visitor.visit(this))
10             return false;
11         return visitor.postVisit(this);
12     }
13
14     public float[] getAffineM(){
15         return null;
16     }
17
18     public void draw(ShaderState curSS){
19
20     }
21
22 }
23
```


VISUAL COMPUTING Lab

SgRootNode

KAIST

```
1  package kr.ac.kaist.vclab.Scenegraph.Node;
2
3  import kr.ac.kaist.vclab.Object3D.RigTForm;
4
5  public class SgRootNode extends SgTransformNode {
6
7      public RigTForm getRbt() {
8          return new RigTForm();
9      }
10
11 }
12
```


VISUAL COMPUTING Lab

SgRbtNode

KAIST

```
package kr.ac.kaist.vclab.Scenegraph.Node;

import kr.ac.kaist.vclab.Object3D.RigTForm;

public class SgRbtNode extends SgTransformNode {
    //SgNode which has a Rigid body transform (RigTForm)

    private RigTForm rbt;

    public SgRbtNode(){
        rbt = new RigTForm();
    }
    public SgRbtNode(RigTForm _rbt){
        rbt = new RigTForm(_rbt);
    }
    @Override
    public RigTForm getRbt(){
        return rbt;
    }

    public void setRbt(RigTForm _rbt){
        rbt = new RigTForm(_rbt);
    }

}
```


VISUAL COMPUTING Lab

SgGeometryShapeNode

KAIST

```
package kr.ac.kaist.vclab.Scenegraph.Node;

import ...

public class SgGeometryShapeNode extends SgShapeNode {
    private  Geometry geo;
    float[] affineM = new float[16];
    float[] color = new float[3];

    public SgGeometryShapeNode(Geometry g, float[] color){
        geo = g;
        System.arraycopy(color, 0,this.color, 0, 3);
        Matrix.setIdentityM(affineM, 0);
    }
    public SgGeometryShapeNode(Geometry g, float[] color, float[] translate, float[] eulerAngles, float[] scales){
        geo = g;
        System.arraycopy(color, 0, this.color, 0, 3);

        float translateM[] = new float[16];
        Matrix.setIdentityM(translateM, 0);
        Matrix.translateM(translateM, 0, translate[0], translate[1], translate[2]);
        float rotateM[] = new float [16];
        Matrix.setIdentityM(rotateM, 0);
        Matrix.setRotateEulerM(rotateM, 0, eulerAngles[0], eulerAngles[1], eulerAngles[2]);

        Matrix.multiplyMM(affineM, 0, translateM, 0, rotateM, 0);
        Matrix.scaleM(affineM, 0, scales[0], scales[1], scales[2]);
    }

    @Override
    public float[] getAffineM(){
        return affineM;
    }

    @Override
    public void draw(ShaderState curSS){
        GLES20.glUniform3fv(curSS.uColorHandle, 1, color, 0);
        geo.draw(curSS);
    }

}
```


VISUAL COMPUTING Lab

RigTForm

KAIST

```
package kr.ac.kaist.vclab.Object3D;

import ...

public class RigTForm extends Object {
    float translateN[];
    Quaternion rotateQ;

    public RigTForm(){
        translateN = new float[]{0,0,0};
        rotateQ = new Quaternion();
    }
    public RigTForm(RigTForm obj){
        float[] temp = obj.getTranslation();
        Quaternion tempQ = obj.getRotation();
        translateN = new float[]{temp[0], temp[1], temp[2]};
        rotateQ = new Quaternion(tempQ);
        temp = null;
        tempQ = null;
    }

    public RigTForm(float[] t, Quaternion r){
        translateN = new float[]{t[0], t[1], t[2]};
        rotateQ = new Quaternion(r);
    }
    public RigTForm(float[] t){
        translateN = new float[]{t[0], t[1], t[2]};
        rotateQ = new Quaternion();
    }
    public RigTForm(Quaternion r){
        translateN = new float[]{0,0,0};
        rotateQ = new Quaternion(r);
    }


    public float[] getTranslation(){
        return translateN;
    }
    public void setTranslation(float[] t){
        translateN[0] = t[0];
        translateN[1] = t[1];
        translateN[2] = t[2];
    }

    public Quaternion getRotation(){
        return rotateQ;
    }
}
```

```
public float[] multiply(float a[]){
    float[] result= new float[4];
    result[0] = translateN[0] + a[3];
    result[1] = translateN[1] + a[3];
    result[2] = translateN[2] + a[3];
    result[3] = 0;
    float temp[] = rotateQ.multiply(a);
    result[0] += temp[0];
    result[1] += temp[1];
    result[2] += temp[2];
    result[3] += temp[3];
    return result;
}

public RigTForm multiply(RigTForm rtf){
    RigTForm result;
    float temp[] = {translateN[0], translateN[1], translateN[2]};
    float temp2[] = {rtf.translateN[0], rtf.translateN[1], rtf.translateN[2], 0};
    float mult[] = rotateQ.multiply(temp2);
    temp[0] += mult[0];
    temp[1] += mult[1];
    temp[2] += mult[2];
    result = new RigTForm(temp, rotateQ.multiply(rtf.getRotation()));
    return result;
}

public static RigTForm inv(RigTForm tForm){
    Quaternion invRot = Quaternion.inv(tForm.getRotation());
    float[] temp = {-tForm.translateN[0],-tForm.translateN[1],-tForm.translateN[2], 1};
    return new RigTForm(invRot.multiply(temp), invRot );
}
public static RigTForm transFact(RigTForm tForm){
    return new RigTForm(tForm.getTranslation());
}
public static RigTForm linFact(RigTForm tForm) { return new RigTForm(tForm.getRotation()); }

public float[] rigTFormToMatrix() {

    float[] tm = new float[16];
    Matrix.setIdentityM(tm,0);
    Matrix.translateM(tm, 0, translateN[0], translateN[1], translateN[2]);
    float[] rm = Quaternion.quatToMat(rotateQ);

    float[] result = new float[16];
    Matrix.multiplyMM(result, 0, tm, 0, rm,0);

    return result;
}
}
```

Lab

Task (1/3)

KAIST

- Create another object using SceneGraph.
- Requirements
  - Move the robot to the left using ***RigTForm.***
  - Make a new object on the right side of the robot
    - make a new Class (e.g., MyRabbit.java)
    - a rabbit, a horse, an alien, etc.

  - Your new object should consist of at least 10 pieces of ***SgGeometryShapeNode***.
  - Your new object should consist of at least 2 kinds of geometry shapes
    - E.g., a **sphere** and a **cube**, just like the robot
  - Your new object may have different colors for each part geometry. (optional)
    - E.g., red arm and blue leg and green head, …

VISUAL COMPUTING Lab

Task (2/3)

KAIST

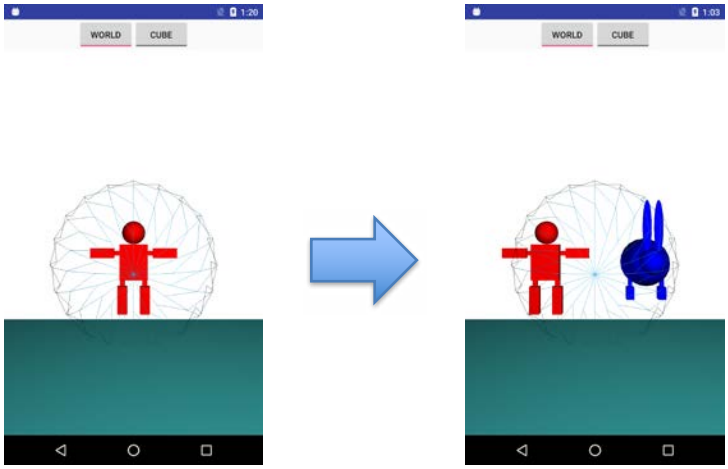
MyGLRenderer.java

```
189     public void initScene(){
190         g_world = new SgRootNode();
191
192         g_skyNode = new SgRbtNode(new RigTForm(new float[]{0f,1f, 10.0f}));
193
194         g_groundNode= new SgRbtNode();
195         //public SgGeometryShapeNode(Geometry g, float[] color, float[] translate, float[] eulerAngles
196         g_groundNode.addChild(new SgGeometryShapeNode(new Square(), new float[]{0.3f,0.9f, 0.9f},
197
198         g_robot1Node =robotFactory.constructRobot(new float[]{1f, 0f, 0f});
199         //Task1: Move the robot to the left
200         //Put your code here
201
202         g_world.addChild(g_skyNode);
203         g_world.addChild(g_groundNode);
204         g_world.addChild(g_robot1Node);
205
206         //Task2: Make a new object on the right side of the robot
207         //Put your code here
208
209         g_currentCameraNode = g_skyNode;
210     }
```


VISUAL COMPUTING Lab

Task (3/3)

KAIST



VISUAL COMPUTING Lab