MSPEL Föreläsning 5

3D–programmering

DSV Peter Mozelius

Den tredje dimensionen

- Babylonierna hade byggnadsritningar som stentavlor redan för ca 4000 år sedan
- Grekerna vidareutvecklade dessa grafiska idéer vidare för ca 2000 år sedan
- Men att uttrycka detta matematiskt kom igång först under renässansen
- Det vi fortfarande använder för datorgrafik formulerades av René Descartes

Den tredje dimensionen

- *Cartesiska koordinater*
- X- , Y- och Z-koordinater
- Projicering av 3D-objekt på 2D-ytor som exempelvis en datorskärm
- I datorer behöver vi även modeller för en realistisk återgivning av ljus och skuggor i våra 3D-världar

Ljusmodeller i 3D-världar

Två *ljusmodeller* för 3D-grafik

1.) **Ray Tracing** eller **strålspårning**

En beräkningsmodell för hur varje pixel i en 3D-värld påverkas av en ljuskälla

2.) **Radiosity** eller **radiositet**

En modell som ger en mer realistisk återgivning men som är beräkningsintensiv

4

Skuggmodeller

- Ju mer processorkraft desto finare och mer realistisk skuggåtergivning
- I *Java 3D API* väljer vi skuggåtergivning med hjälp av konstanterna:
  - **SHADE_FLAT**
  - **SHADE_SMOOTH**
  - **SHADE_GOURAUD**

5

Ljuskällor

- *Ambient light*
  - Samma intensitet åt alla håll och på samtliga ställen. För att beskriva hur ljus reflekteras från objekt.
- *Directional light*
  - För ljus från en avlägsen ljuskälla med enbart en riktning (som solen)
  - Konstant ljusvektor

6

Ljuskällor

- Ljuskällor med icke-konstant ljusvektor där
- **ljusintensiteten är beroende av avståndet**
- *Point light* eller punktljus
  - En modell för att efterlikna glödlampor, stearinljus mm, källan har en position
- *Spot light*
  - en specialisering av Point light

7

Z-buffer algoritmen

- 3D-rendering är beräkningsintensiv
- Behov av att effektivisera renderingen
- Man försöker undvika att räkna på pixlar i objekt som är skymda i Z-led
- Algoritmer för **Hidden Surface Removal**
- På svenska: *Borttagning av skymda ytor*
- Vanligast är: **Z-buffer algoritmen**

8

Java 3D API

- Java 3D API en *standard extension* som har blivit ett samarbetsprojekt: https://java3d.dev.java.net/
- Hämtas hem från:
  - `http://java.sun.com/javase/technologies/desktop/java3d/`
  - OpenGL (mest oberoende)
  - Direct3D (snabbare i Windows)
  - QuickDraw3D (Mac)

Paus 15 min ?

9

Java 3D API

- De två grundläggande paketen är:
- **javax.vecmath**
  - med stöd för vektorbaserad matematik
- **com.sun.j3d.utils**
  - med över hundra klasser för:
  - *3D-primitiver* som kuber och sfärer
  - Grundläggande 3D-operationer

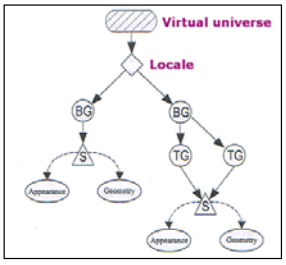
10

Java 3D API

- Vid en installation av Java 3D API så läggs bl a följande filer in:
  - **vecmath.jar**
  - **j3dcore.jar**
  - **j3dutils.jar**
  - **j3daudio.jar**

11

Java 3D API



- **Simple Universe**
  - Ett enkelt litet universum som inkluderar ett **Locale** objekt

12

Java 3D API

- I *Locale* finns det koordinatsystem där enheten 1.0f ska motsvara 1 meter
- API:t har en strikt hierarki och det är endast objektet *BranchGroup* som kan hängas in under en Locale
- Med en kombination av Branchgroups modulariserar du dina 3D-världar

13

Java 3D API

- Under en BranchGroup går det sedan att lägga in en *TransformGroup*
- En TransformGroup är en speciell nod i API:t som möjliggör saker som **rotation**, **förflyttning** och **skalning** av 3D-objekt
- En TransformGroup arbetar med matematiska matrisberäkningar

14

Java 3D API

- Klassen **Shape3D**
  - Definierar visuella objekt
  - Är en klass av typen Leaf
- Ett Shape3D-objekt består av:
  - *Geometry*
  - *Appearance*

```
Shape3D(), Shape3D(Geometry g)
Shape3D(Geometry g, Appearance a)
```

15

Java 3D API

- Java 3D primitiver finns i
  - `com.sun.j3d.utils.geometry`
- **Box**();
- **Cone**(float radie, float hojd);
- **Cylinder**(float radie, float hojd);
- **Sphere**(float radie, Appearance app);
-

Färg eller ytmaterial i ett **Appearance**

16

Java 3D API

Klassen Appearance styr renderingen
Sätt färg på ett objekt med hjälp av
en instans av Appearance:

```
Appearance app = new Appearance();
ColoringAttributes ca;
ca = new ColoringAttributes();
ca.setColor(R, G, B);
app.setColoringAttributes(ca);
```

17

Java 3D API

- Att rotera ett 3D-objekt görs med:
  - 1.) En **RotationInterpolator**
  - 2.) Ett **Alphaobjekt**

```
new RotationInterpolator(
      Alpha a, TransformGroup tg);
```

Alphaobjektet synkroniserar rotationstiden

18

Java 3D API

- Om du har en *RotationInterpolator* och t ex vill få den att snurra i X-led så gör detta med en Transform3D:

```
Transform3D rotor;
rotor = new Transform3D();
rotor.rotX(double vinkel);
```

- Vinkeln ska anges i **radianer**

19

Java 3D API

- För att öka realismen i 3D-världarna så arbetar man med:
  - **textures** (engelska)
  - **ytmaterial** (svenska)
- Om ett fotorealistiskt ytmaterial först skapas som en JPEG-bild **metall.jpg**
- Så kan det sedan laddas in genom:

```
TextureLoader("metall.jpg","RGB",this);
```

20

Java 3D API

- Det finns två olika sätt att lägga in text
- 1.) Klassen **Text2D**
  - Uppbyggd av polygoner som kan kläs med ytmaterial
- 2.) Klassen **Text3D**
  - Skapar tredimensionell grafik med hjälp av en instans av **FontExtrusion**

Paus

21

Äpplen som Applikationer

- I paketet **`com.sun.j3d.utils.applet`** finns klassen **`MainFrame`**
- Din kod blir körbar som både Applet och Applikation genom:

```
public static void main(String[] args){
   Frame f = new MainFrame(
            new Klass(), bredd, hojd);
}
```

22

Grader - Radianer

- 1 radian är ett cirkelsegment som är lika långt som cirkelns radie:
  - 1 varv = **2 * π radianer** = 6.283 radianer
  - 180 grader = **π radianer** = 3.142 radianer
  - 90 grader = **π/2 radianer** = 1.571 radianer
  - 45 grader = **π/4 radianer** = 0.785 radianer

23

Interaktion i 3D-världarna

- Ett enkelt sätt för musinteraktion är med `com.sun.j3d.utils.behaviors.mouse.*;`

  Användbar kod för att kunna göra lite annorlunda lösningar av Uppgift4g:

```
MouseRotate rotera = new MouseRotate();
 ...
minTransformGroup.addChild(rotera);
```

24

Ett enkelt komigångexempel

```
import java.awt.*;
import java.applet.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.applet.MainFrame;

public class Xempel3D extends Applet{
```

25

Ett enkelt komigångexempel

```
public void init(){
  setSize(400,300);
  setLayout(new BorderLayout());

  GraphicsConfiguration gConfig =
   SimpleUniverse.getPreferredConfiguration();

  Canvas3D c3D = new Canvas3D(gConfig);
  add("Center", c3D);

  SimpleUniverse su = new SimpleUniverse(c3D);
```

26

Ett enkelt komigångexempel

```
//Initiera till ett passande betraktningsavstånd
su.getViewingPlatform().
setNominalViewingTransform();

//Lägg till en BranchGraph med en färgad kube
su.addBranchGraph(
      makeBranchGraphWithColorCube());

}//init
```

27

Ett enkelt komigångexempel

```
//Skapa en BranchGroup med en färgad kub
BranchGroup makeBranchGraphWithColorCube(){
   BranchGroup bg = new BranchGroup();
   ColorCube cc = new ColorCube(0.5);
   bg.addChild(cc);
   return bg;
}//makeBranchGraphWithColorCube
```

28

Ett enkelt komigångexempel

```
//det ska även gå att köra som applikation
 public static void main(String args[]){
   new MainFrame(new Xempel3D(), 400, 300);
 }
}//klass Xempel3D
```

Tack för idag!

29