Special effects: Fog

Fog: Similar effect like attenuation.

Atmospheric attenuation: Light is absorbed (for instance by dust in the air).

Fog: Diffuse reflection occurs on tiny drops of water in the air.

Objects in the distance are less visible with fog and attenuation.

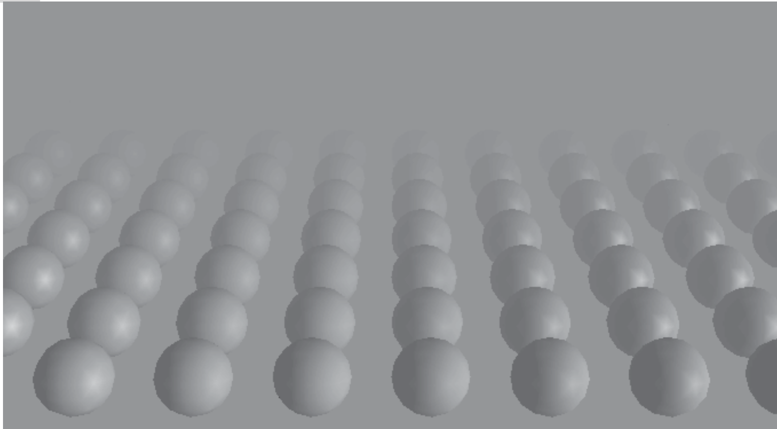
In contrast to atmospheric attenuation, fog also causes a white or grey background colour.

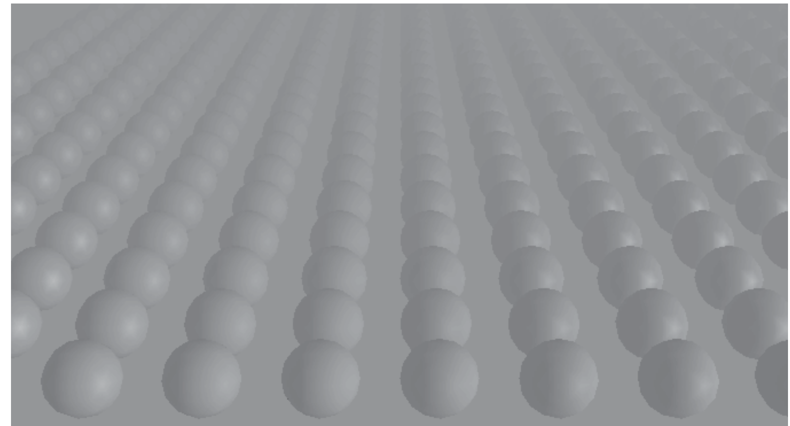
The colour of an object is blended with the colour of the fog.

Attenuation corresponds to black fog.

© Springer

Fog





Special effects: Fog

Blending of the fog colour and the object's colour:

$$b(d) \cdot I_{\text{fog}} + (1 - b(d)) \cdot I_{\text{object}}.$$

- I_{fog}: (Colour) intensity of the fog.
- I_{object}: (Colour) intensity of the object.
- d: Distance of the object to the viewer.
- $b : \mathbb{R}_0^+ \to [0,1]$ increasing blending function with $b(0) = 0$ and $\lim_{d\to\infty} b(d) = 1$.

© Springer

Special effects: Fog

Linear fog: No blending with the fog up to distance $d = d_0$, 100% fog starting at distance $d = d_1$, linear blending in between:

$$b(d) \ = \ \begin{cases} 0 & \text{if } d \leq d_0, \\ \frac{d-d_0}{d_1-d_0} & \text{if } d_0 < d < d_1, \\ 1 & \text{if } d_1 \leq d. \end{cases}$$

Exponential fog: Exponential increase of the effect of fog controlled by the (density) factor $\alpha > 0$:

$$b(d) \ = \ 1 - e^{-\alpha \cdot d}$$

© Springer

Java 3D: Fog

Linear fog in Java 3D:

```
LinearFog fog =
          new LinearFog(colour,d_0,d_1);

fog.setInfluencingBounds(bounds);

theScene.addChild(fog);
```

Analogously for exponential fog:

```
ExponentialFog fog =
      new ExponentialFog(colour,alpha);
```

© Springer

Java 3D: Fog

It is recommended to adjust the background colour to the colour of the fog.

A background image will not be blended with fog.

(see `ExponentialFog` and `LinFogExample.java`)

© Springer

Java 3D: Links

The class `Link` is used in the programm `FogExample.java` in order to use an object or a part of a scene more than once in the same scene.

Example: A room with a number of doors of identical type.

Let `tgDoor` be the transformation group containing the generic model of the door.

Definition of a `SharedGroup`:

```
SharedGroup sgDoor = new SharedGroup();
sgDoor.addChild(tgDoor);
```

© Springer

Java 3D: Links

Then the door can be used in the scene in different places as a `Link`, for instance:

```
tgDoor1.addChild(new Link(sgDoor));

tgDoor2.addChild(new Link(sgDoor));

...
```

`tgDoor1` and `tgDoor2` are transformation groups placing the door in different positions.

© Springer

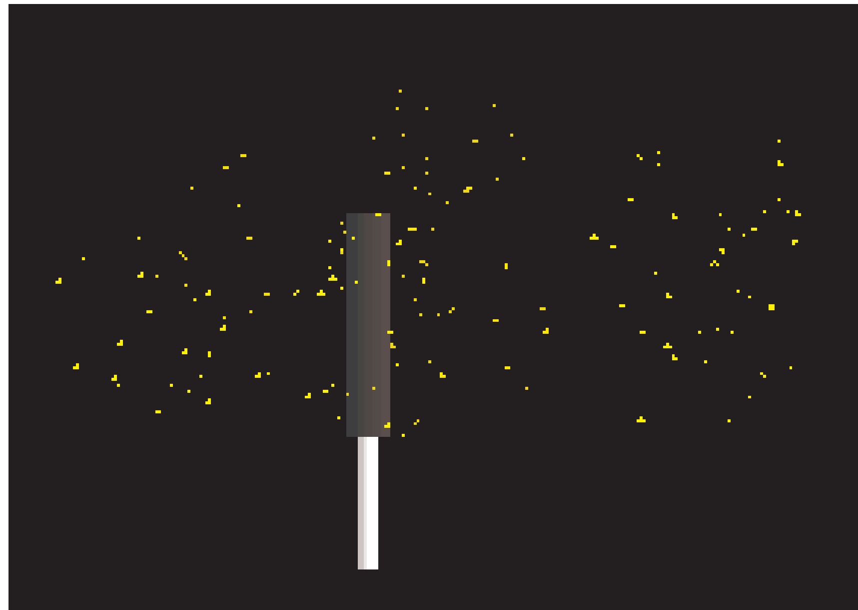
Particle systems

Examples for particle systems: Wafts of fog, explosions, fire, fountains.

- Many small particles that are not controlled individually, but by a few parameters.
- A random mechanisms is involved determining the parameters for the single particles.
- Basic behaviour pattern for all particles: random life span, random direction of movement, random splitting,. . .

© Springer

Particle systems



Control parameters of particle systems

- Position where a particle is generated.
- Initial velocity of a particle.
- Direction in which the particle is moving.
- Life span of a particle.
- Intensity with which particles are emitted or generated.
- How a particle should look like.
  often: particle = set of small elements,
  e.g. particle = square with a suitable texture.

© Springer

Simple particle system in Java

- Transformation group `tgParticleSystem` to which the particle system is assigned.

```
tgParticleSystem.setCapability(
   TransformGroup.ALLOW_TRANSFORM_WRITE);
tgParticleSystem.setCapability(
   TransformGroup.ALLOW_CHILDREN_EXTEND);
```

© Springer

Simple particle system in Java

- Waiting time specified in milliseconds in the form of a `long`-value, defining when the particle system should start to emit particles.
- Duration in milliseconds in the form of a `long`-value how long the particle system should run.
- Particle generator (Interface `ISimpleParticleCreator`) Method `getNextParticle`
- Waiting time betweeen the creation of two succeeding particles (Interface `ILongCreator` Method `getNextLong`

Simple particle system in Java

- Live span of a particle (Interface `ILongCreator`)
  Method `getNextLong`
- Velocity of a particle (Interface `IFloatCreator`)
  Method `getNextFloat`
- Initial position of a particle (Interface `IVector3fCreator`)
  Method `getNextVector3f`
- Direction of movement of a particle (Interface `IDirectionCreator`)
  Methods: `getNextZRotation`, `getNextZRotation`

© Springer

Simple particle system in Java

- `activateParticleSystem` starts the `Thread` `SimpleParticleSystemRunner`.
- In this class: Creation of a `BranchGroup` `bgParticleSystem` to which the single particles are assigned.

The child nodes of this `BranchGroup` are changed dynamically during runtime.

© Springer

Dynamic change of the scenegraph

```
bgParticleSystem.setCapability(
        BranchGroup.ALLOW_DETACH);
bgParticleSystem.setCapability(
      BranchGroup.ALLOW_CHILDREN_WRITE);
bgParticleSystem.setCapability(
      BranchGroup.ALLOW_CHILDREN_EXTEND);
bgParticleSystem.setCapability(
      BranchGroup.ALLOW_CHILDREN_READ);
```

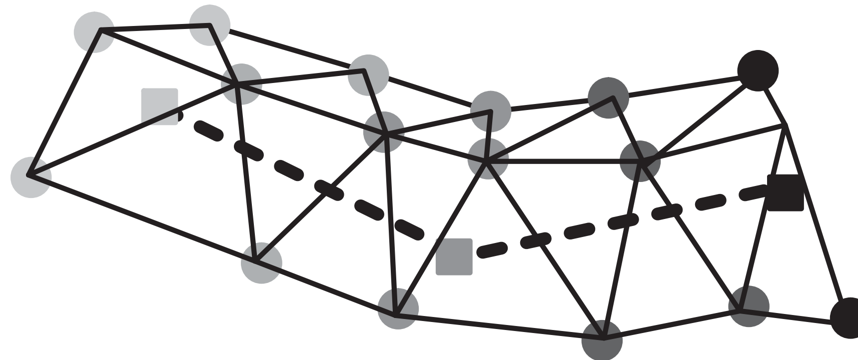
Remove a node from the scenegraphen:
`removeChild`

© Springer

Dynamic surfaces

Nonrigid objects: Skin, cloth,. . .

Modelling with skeletons and skinning.



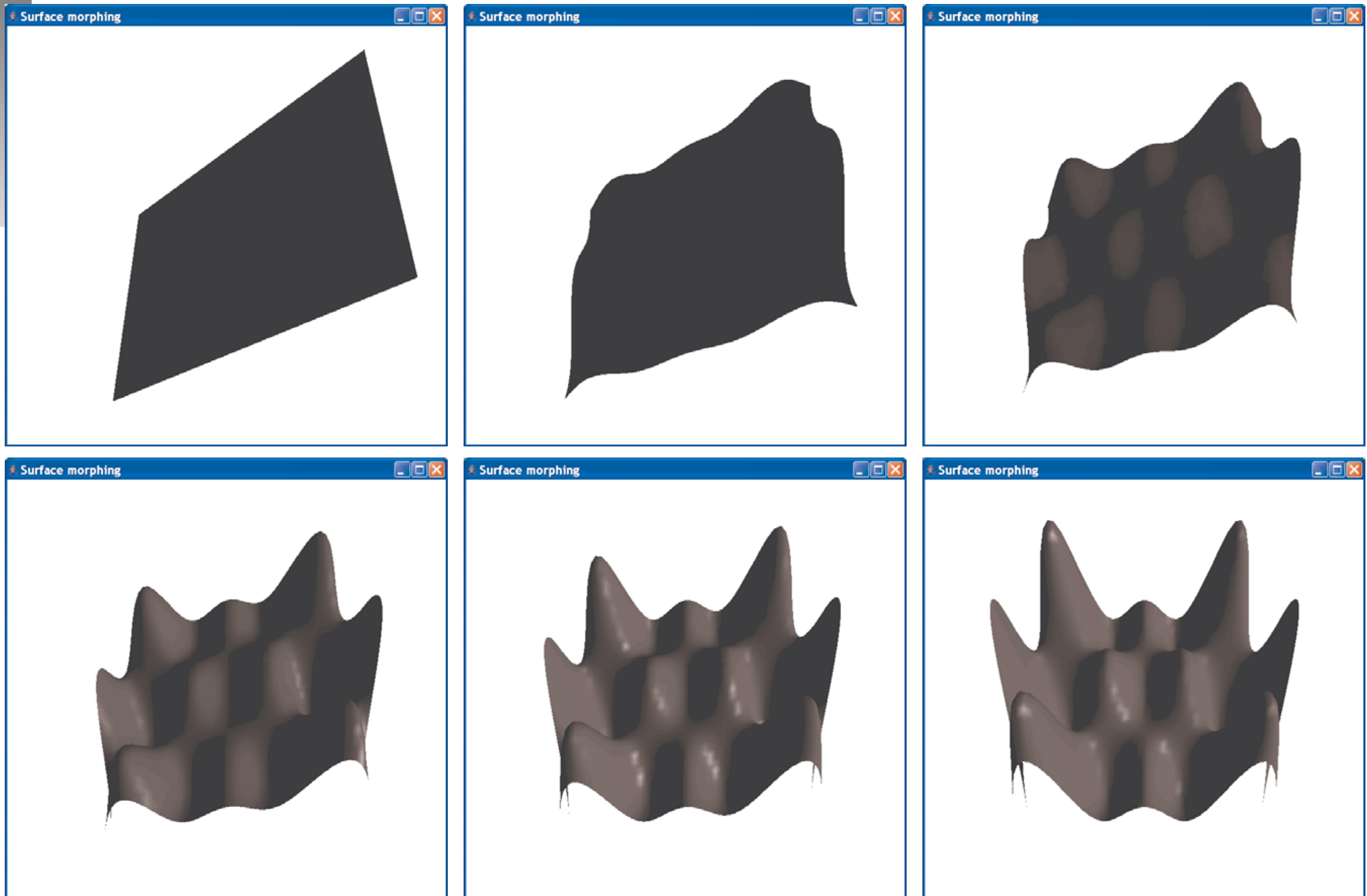
Dynamic surfaces

- A few inner skeleton points $\mathbf{s}_i$ to which transformations T_i are applied.
- The vertex points $\mathbf{p}$ at the surface are assigned to the skeleton points using weights $w_i^{(\mathbf{p})}$.
- The transformation applied to the vertex point $\mathbf{p}$ is then given by

$$T_{\mathbf{p}} = \sum_i w_i^{(\mathbf{p})} \cdot T_i.$$

The weights $w_i^{(\mathbf{p})}$ must be normalised, so that $T_{\mathbf{p}}$ is a convex combination of the T_i.

Dynamic surfaces



© Springer

alternative model:

Interpolation between surfaces that are modelled by corresponding triangles.

Implementation example in `DynamicSurfaceExample`: Interpolation between two `GeometryArray`s defined by functions in two variables.

© Springer

Dynamic surfaces

Use of a `Morph` object:

Node ina scenegraph to which an array of `GeometryArray` objects and an array of weights for the interpolation is assigned.

dynamic change of the weights by a `Behavior`.

Dynamic surfaces

```
public class SimpleMorphBehaviour extends Behavior
{
   private Morph theMorph;
   private Alpha theAlpha;

   private double[] weights;
   private WakeupCondition trigger =
                             new WakeupOnElapsedFrames(0);

   public SimpleMorphBehaviour(Morph targetMorph, Alpha alpha)
   {
     theMorph = targetMorph;
     theAlpha = alpha;
     weights = new double[2];
   }
```

© Springer

Dynamic surfaces

```
    public void initialize()
    {
      wakeupOn(trigger);
    }

    public void processStimulus(Enumeration criteria)
    {
      weights[0] = 1 -  theAlpha.value();
      weights[1] = 1 -  weights[0];

      theMorph.setWeights(weights);
      wakeupOn(trigger);
   }
}
```

© Springer

Interaction: Picking objects

The `PickMouseBehavior` enables the user to select objects with the mouse and to initiate actions.

First step: Definition of a new class extending `PickMouseBehavior`.

The actions to be carried out when objects are picked are defined inside this extended class.

Interaction: Picking objects

```
MyPickingBehaviour extends
                   PickMouseBehavior

public MyPickingBehaviour(Canvas3D
                               pCanvas,
                   BranchGroup root,
                   Bounds pBounds,
                     ...)
{
  super(pCanvas,root,pBounds);
  setSchedulingBounds(pBounds);
  ...
}
```

© Springer

Interaction: Picking objects

`pCanvas` must be the `Canvas3D` on which the scene is displayed.

`root` should be the `BranchGroup theScene`.

`pBounds` defines the bounding region in which it is possible to pick the objects.

Further parameters might be needed to control the actions to be initiated when an object is picked.

© Springer

Interaction: Picking objects

Inside the class `MyPickingBehaviour`, the method

```
public void updateScene(int xpos,
                        int ypos)
```

must be overwritten. Inside this method: Determine which object has been picked:

```
pickCanvas.setShapeLocation(xpos,ypos);

PickResult pResult =
              pickCanvas.pickClosest();
```

© Springer

Interaction: Picking objects

Since clicking on the two-dimensional projection might not always refer to a unique object in 3D, the method `pickClosest()` is used here.

The method `pickAll()` is also available, returning an array containing all objects projected to the point which is clicked.

© Springer

Interaction: Picking objects

The chosen objects should either be elementary objects of the class `Primitive` or an instance of the class `Shape3D`.

It is also possible to define a pickable region by defining a completely transparent sphere, which is not visible, but pickable.

In order to identify which object has been picked in the scenegraph, the method `setUserData(...)` (for example `setUserData("’nodeName"’)`) allows to assign an arbitrary Java object (here, a string) to a node of the scenegraph.

© Springer

Interaction: Picking objects

The `PickResult` must be casted into an object of the type `Primitive` or `Shape3D`, for instance

```
Primitive pickedShape = (Primitive)
  pResult.getNode(PickResult.PRIMITIVE);
```

Then it is possible, to identify the picked object by calling the method `pickedShape.getUserData()`.

© Springer

Interaction: Picking objects

The instance of the class `MyPickingBehaviour` must be added to the scene by `addChild(...)`.

When interpolators should be started after objects have been picked, the corresponding `Alpha`s should be passed to `MyPickingBehaviour`. The `StartTime` should initially set to “infinity” and inside the method `updateScene` to “now”.

(see `InteractionTest.java`/`PickingTest.java` and `PickingExample.java`/`InteractionExample.java`)

© Springer

Interaction: Collision detection

Collision detection refers to the problem of determining whether moving objects collide, i.e. whether the corresponding shapes intersect..

Without collision detection objects can penetrate each other.

© Springer

Interaction: Collision detection

Possible strategies:

- Objects are enclosed in bounding volumes (boxes or spheres).

  Collision is checked only on the level of the bounding volumes.

  - Advantage: Simple computations for collision detection.
  - Disadvantage: For objects with complex shapes a collision might be indicated although the actual objects are not touching each other.

© Springer

Interaction: Collision detection

- Collision detection based on surface polygons
  - Advantage: Collision corresponds (almost) exactly to touching objects.
  - Disadvantage: High computational costs.

Mixed strategy: First apply collision detection based on bounding volumes and in case of a collision of the bounding volumes, check collision on the level of polygons.

© Springer

Interaction: Collision detection

Simple bounding volumes:

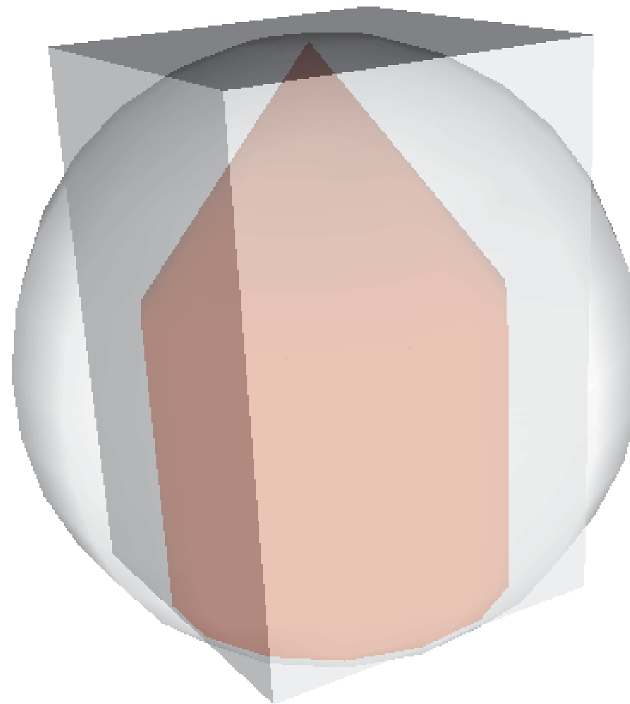
- Axes-parallel boxes:
  - Smallest bounding volume is easy to compute.
  - Collision detection becomes more complicated when the objects move and the bounding volumes are no longer axes-parallel.
- Spheres:
  - Smallest bounding volume not easy to compute.
  - Collision detection is very simple. Test for

Radius_1 + Radius_2 $\geq$ distance of the midpoints

© Springer

Interaction: Collision detection

Bounding box and bounding sphere for an object



© Springer

Java 3D: Collision detection

Definition of a class `CollisionBehaviour`, extending the class `Behaviour`.

```
public class CollisionBehaviour extends
                                     Behavior
{
   private WakeupOn...;
   ...
}
```

Different collision criteria can be defined for initiating a change:

© Springer

Java 3D: Collision detection

- `WakeupOnCollisionEntry`: At the beginning of a collision.
- `WakeupOnCollisionExit`: After a collision has been resolved.
- `WakeupOnCollisionMovement`: During the collision (during a movement).
- `WakeupOr`: A combination of the previous criteria.

Further attributes might be needed in the class `CollisionBehaviour` for parameters to control the actions to be initiated in case of a collision (similar to `MyPickingBehaviour`).

© Springer

Java 3D: Collision detection

First, the method `initialize()` must be overwritten, for example:

```
public void initialize()
{
 hit = new WakeupOnCollisionEntry(node);
 wakeupOn(hit);
}
```

when an action should be initiated at the beginning of a collision with an object of the node `node` in the scenegraph.

`hit` must be defined as a corresponding attribute in the class `SimpleCollision`.

© Springer

Java 3D: Collision detection

When an action should be initiated after a collision is resolved, use

```
public void initialize()
{
  criteria = new WakeupCriterion[2];

  criteria[0] = new
           WakeupOnCollisionEntry(node);

  criteria[1] = new
            WakeupOnCollisionExit(node);

  oredCriteria = new WakeupOr(criteria);
  wakeupOn(oredCriteria);
}
```

© Springer

Java 3D: Collision detection

```
public void processStimulus(
                        Enumeration enum)
{
  while (enum.hasMoreElements())
  {
    WakeupCriterion criterion =
   (WakeupCriterion) enum.nextElement();
    if (criterion instanceof
                WakeupOnCollisionEntry)
    {
      ...
    }
    wakeupOn(hit);
  }
}
```

© Springer

Java 3D: Collision detection

In . . . the corresponding action is initiated that should be carried out when the collision occurs.

An instance of the corresponding class must be generated and added to the scene:

```
CollisionBehaviour cb =
          new CollisionBehaviour(...);

theScene.addChild(cb);
```

(see `CollisionExample.java`)

© Springer

Java 3D: PickTranslateBehavior

Java 3D provides a class for moving objects with mouse, pressing the right mouse button.

```
PickTranslateBehavior pickTrans =
  new PickTranslateBehavior(theScene,
                            myCanvas3D,
                            bs);

theScene.addChild(pickTrans);
```

`bs` is a `BoundingRegion`.

© Springer

Java 3D: PickTranslateBehavior

Certain attributes must be set for the corresponding transformation groups:

```
tg.setCapability(
  TransformGroup.ALLOW_TRANSFORM_WRITE);

tg.setCapability(
  TransformGroup.ALLOW_TRANSFORM_READ);

tg.setCapability(
  TransformGroup.ENABLE_PICK_REPORTING);
```

© Springer

Java 3D: Switches

A `Switch` is a node in the scenegraph with a number of child from which one (or a subset) can be selected for display in the scene.

In this way, one object can be changed into another one.

```
Switch sw = new Switch();

sw.setCapability(
            Switch.ALLOW_SWITCH_WRITE);

sw.addChild(tg0);

sw.addChild(tg1);
...
```

© Springer

Java 3D: Switches

The method `setWhichChild(childNumber)` allows to choose which of the child nodes should be included in the scene.

Using the class `BitSet` (in `java.util`) is possible to define a bitmask `bm`.

The methods `bm.set(childNumber)` and `bm.clear(childNumber)` and then `sw.setChildMask(bm)` allow the choice of an arbitrary subset of the child nodes of the `Switch` to be included in the scene.

© Springer

Java 3D: Keyboard navigation

- Definition of a transformation group `tgAll`.
- The objects in the scene are not assigned directly to `theScene`, but to `tgAll`.
- `tgAll` is assigned to `theScene`.

```
TransformGroup tgAll =
                   new TransformGroup();
tgAll.setCapability(
   TransformGroup.ALLOW_TRANSFORM_READ);
tgAll.setCapability(
  TransformGroup.ALLOW_TRANSFORM_WRITE);
...
```

© Springer

Java 3D: Keyboard navigation

```
theScene.addChild(tgAll);

KeyNavigatorBehavior knb =
       new KeyNavigatorBehavior(tgAll);

knb.setSchedulingBounds(bounds);

tgAll.addChild(knb);

tgAll.addChild(...);
```

© Springer

Java 3D: Acoustic effects

- `BackgroundSound`: Sound equivalent of ambient light.
- `PointSound`: Sound equivalent of a point light source.
- `ConeSound`: Sound equivalent of a spotlight.

A `Sound` can be assigned to a transformation group or directly to the scene.

© Springer

Java 3D: Acoustic effects

Loading a sound file:

```
MediaContainer medCon;
try
{
  FileInputStream is =
     new FileInputStream("mysound.wav");
  medCon = new MediaContainer(is);
  theSound.setSoundData(medCon);
}
catch(Exception e){...}
```

© Springer

Java 3D: Acoustic effects

Setting the necessary parameters:

```
theSound.setEnable(true);
theSound.setLoop(Sound.INFINITE_LOOPS);
theSound.setInitialGain(0.9f);
theSound.setSchedulingBounds(bs);
```

(see `SoundExample.java`)

© Springer

3D information is obtained from a number of sources.

Distinction between **monocular** (with one eye) and **binocular** (combination of both eyes) factors.

© Springer

Monocular factors

The focus (accommodation) : When the eye views an object, the lens has to be adjusted by muscle contractions in the eye so that the object is in focus. Objects in the far background get out of focus. This provides a certain information about distances.

Parallax of movements : Parallax of movements can be observed when objects move relative to each other. The position and the size of the objects are crucial for this information. Moving objects change their distance to the viewer.

© Springer

Monocular factors

Masking : When one object is hidden partly from sight by another object it can be concluded that the completely visible object must be closer to the viewer than the other object.

Light and shadows : From the location, the direction, the shape and the size of the shadow cast by an object, conclusions can be drawn about the location of the object itself and about the shape of the object on which the shadow is cast.

© Springer

Monocular factors

Size : From the size of an object in an image compared to the size of other objects and the known size of the object, the distance of the object to the viewer can be estimated. Closer objects are larger. This means if an object that is known to be large occurs small, it must be far away from the viewer.

Attenuation : The effects of atmospheric attenuation increase with the distance. Objects in the far distance appear less contrasted than closer ones.

Head movements : Moving the head generates different images (over time) from which 3D information can be extracted.

© Springer

Binocular factors

Difference of the images : Since the two eyes view the same scene from two slightly different positions, 3D information can be retrieved from a comparison of the two images.

Convergence : The eyes can modify the direction of view by turning the eyeballs slightly. The closer an object is, the more the eyeballs are turned inside. Based on this slight turn, the distance of an object in focus can also be roughly estimated.

Pulfrich effect : A bright stimulus is processed faster by the brain than a dark one. Wearing glasses with one darkened side, this effect can be used to generate 3D impressions for special image sequences.

© Springer

Generating stereoscopic images

Anaglyph images are a very old approach to stereoscopic viewing. The two images are drawn with different colours, in most cases one in red and the other one in green. To view the two overlaid images special glasses must be worn with different colour filters for the two eyes. A disadvantage of anaglyph images is the loss of colour information.

© Springer

Generating stereoscopic images

Polarised light is a better alternative. Light waves oscillate around the axis of the direction in which they spread. Polarised light oscillates only in one plane. The images for the two eyes are projected to a screen with different polarisations. The viewer must wear glasses with the corresponding polarisations for the two eyes.

© Springer

Generating stereoscopic images

Liquid crystal shutter glasses: The images for the two eyes are presented alternatingly on the computer screen. The user must wear liquid crystal shutter glasses which are synchronised with the monitor by an infrared signal.

Since the image frequency for each eye is reduced by half in this way, it is recommended to use a special monitor with a higher frequency of at least 100 Hz. The frequency is also limited by the phosphorescence effect of monitors.

© Springer

Generating stereoscopic images

Head-mounted display: Usually a small helmet with two LCD displays with magnifying lenses, one for each eye. Head tracking is required when the viewer can move around in the scene wearing the helmet. Headphones can also be integrated into the helmet for sound effects.

The disadvantage is that wearing a helmet is not as comfortable as wearing glasses.

© Springer

Generating stereoscopic images

Specific displays that do not need specific glasses for the separation of the images for the two eyes. Some techniques use a specific mask of lenses or prisms in front of the display so that each eye can only see one half of the pixels.

3D effect is visible only in a small range.

Holographic techniques sometimes based on rotating projection surfaces.

© Springer

Parallax

negative parallax: The object is located in front of the screen (projection plane).

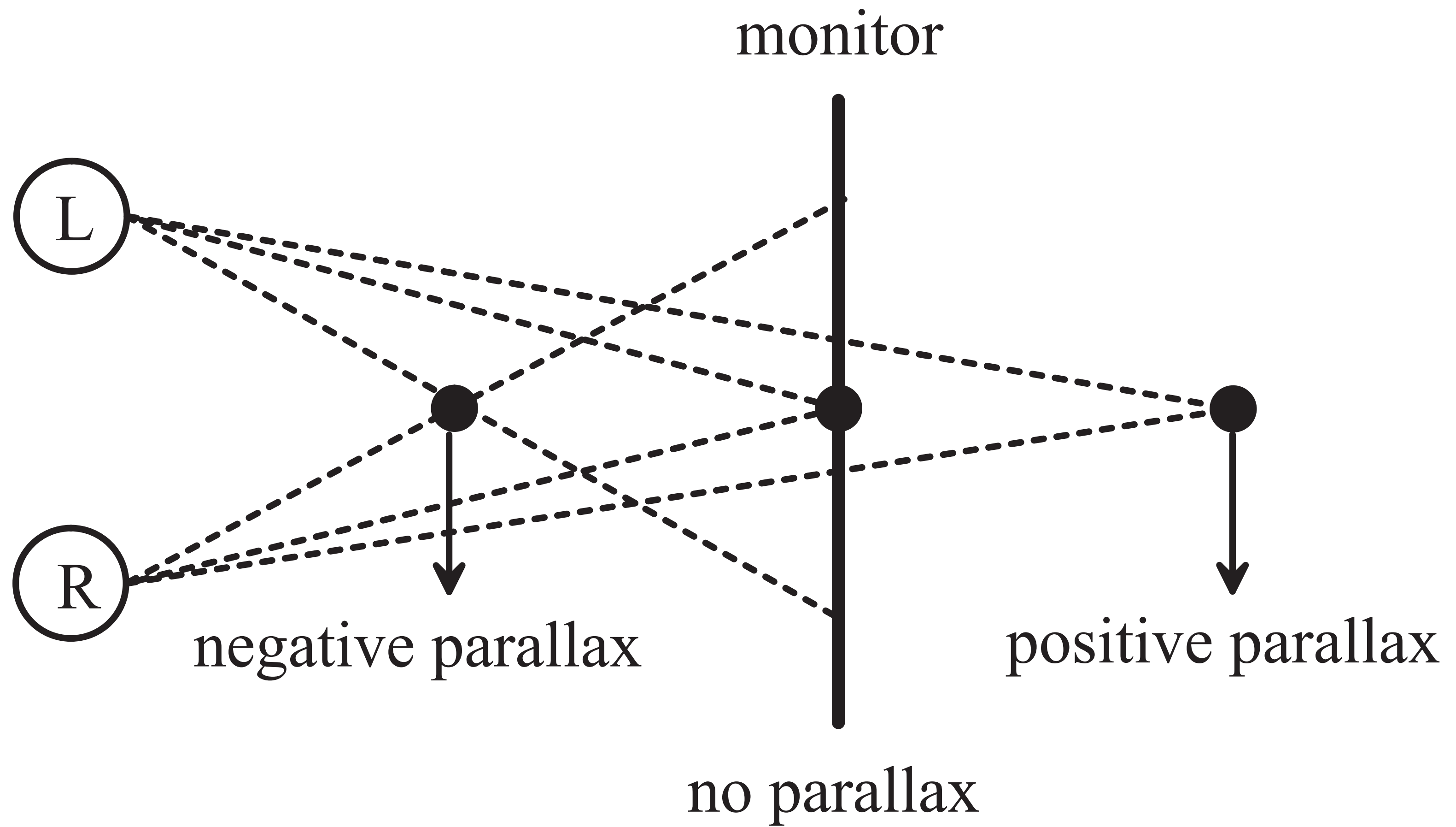
no parallax: The object is located in the projection plane.

positive parallax: The object is located behind of the projection plane.

divergent parallax: Contradictory position showing an object left of the left eye and right of the right eye which is impossible in reality.

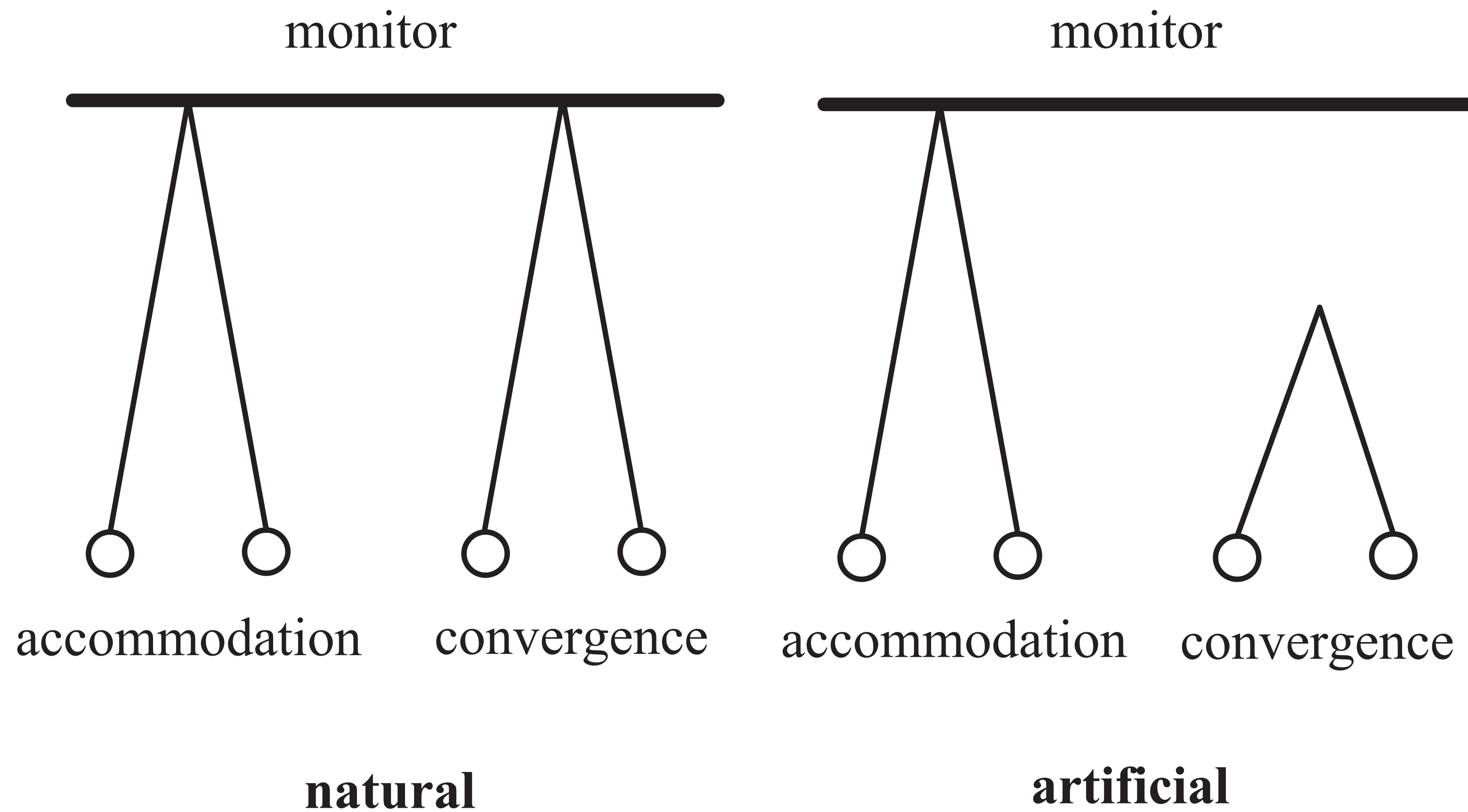
© Springer

Parallax



© Springer

Parallax



© Springer

Undesired effects

- Positive parallax: Seeing the object through a window (the monitor).
- Negative parallax: Objects are simply cut off at the edge of the monitor.
- Very strong parallax: Crosstalk.
- Ghosting effects for slower monitors.
- Avoid divergent parallaxes.

© Springer