

# Wassersimulation (Wellenmodell)

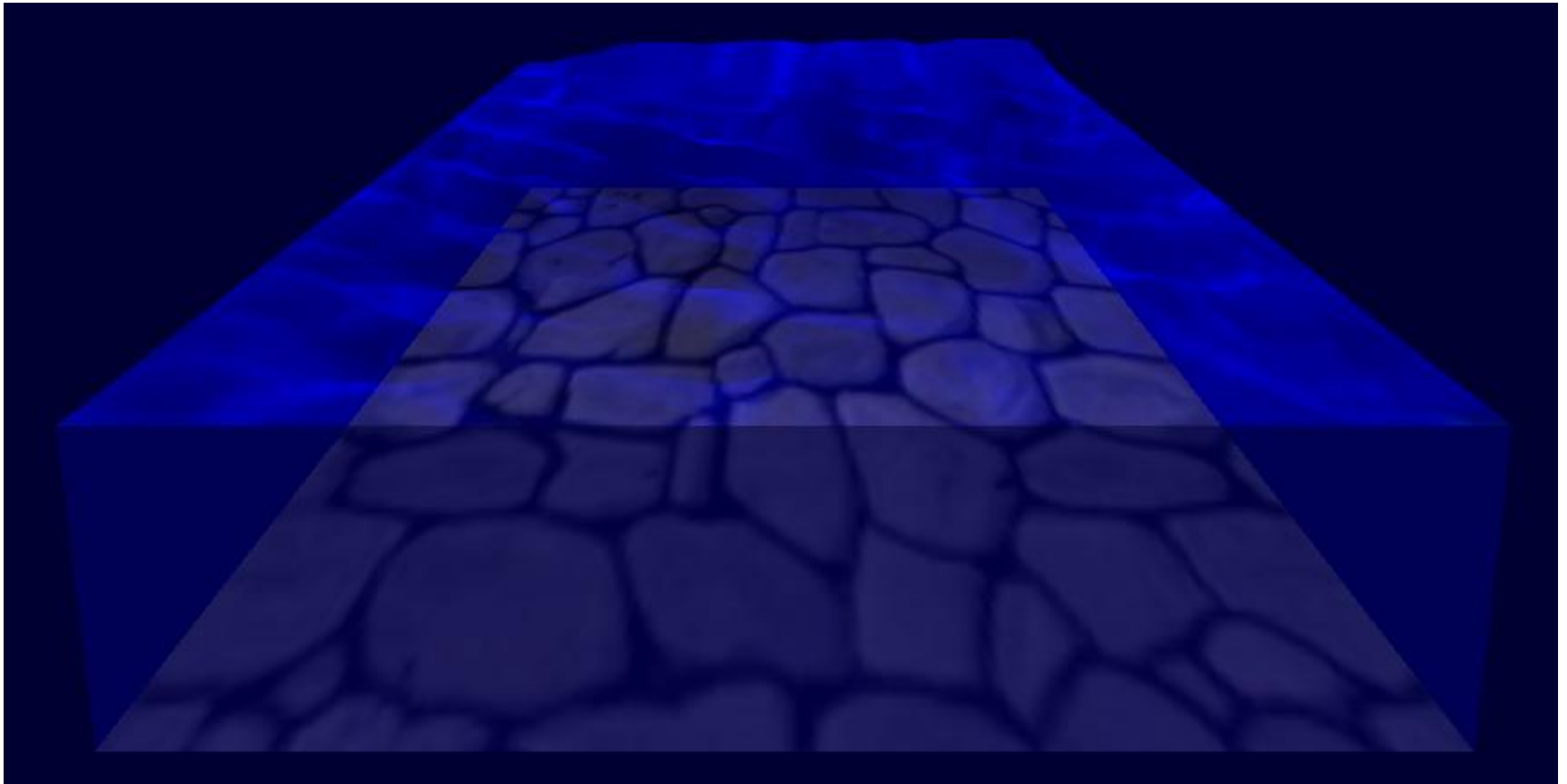
Martin Groher

03 July 2006

chair for computer aided medical procedures

department of computer science | technische universität münchen

# Wassersimulation



# Inhalt

- WaterManager
  - Datentypen
  - Node Erstellung
  - Update (Wellenbewegung)
  - Disturb (Wellenmaschine)
  - Normalen
  - Texturen / Rendern

# Datentypen

- `std::vector< std::vector<Point3D*> >`
- `Point3D`
  - `Position (Vector3D)`
  - `Normale (Vector3D)`
  - `Farbe (Color)`
- `Hilfswerte`
  - `startPoint (Vector3D)`
  - `Breite (uint width) – Mit Rand: 1 weniger als im vector`
  - `Länge (uint length) – Mit Rand: 1 weniger als im vector`
  - `Dichte (float density)`
  - `Dämpfung (float dampFactor)`

# Node Erstellung

- `std::vector< std::vector<Point3D*> >`
  
- `for (unsigned int i = 0; i < length + 1; ++i)`
  - `std::vector<Point3D*> temp;`
  - `nodes.push_back(temp);`
  - `for (unsigned int j = 0; j < width + 1; ++j)`
    - `Point3D* pointToAdd = new Point3D();`
    - `nodes[i].push_back(pointToAdd);`

# Update - Wellenbewegung

- In der Vorlage werden die Wellen pro Zeile nach Zeit aktualisiert
  
- Der Algorithmus arbeitet auf einem Datensatz
  - Jede Berechnung basiert auf allen vorigen Änderungen
  - Optimiert auf Wellen
  - Nicht geeignet für Teichsimulationen

# Disturb – Wellenmaschine

- In der letzten Reihe sollte alle *wManagerNewWave* Sekunden die Höhe um *wManagerWaveGeneratorHeight* erhöht werden
  
- ```
for (unsigned int i = 1; i < width; ++i)
```

  - ```
nodes[length-1][i]->setPosition(Vector3D(
```

    - ```
nodes[length-1][i]->getPosition().x,
```
    - ```
nodes[length-1][i]->getPosition().y + WAVE_GENERATOR_HEIGHT,
```
    - ```
nodes[length-1][i]->getPosition().z)
```
  - ```
);
```

# Normalen

- Ansatz:
  - Kreuzprodukt aus zwei Vektoren:
  - $\text{Vector3D } a = \text{Vector3D}[i-1][j] - \text{Vector3D}[i+1][j]$
  - $\text{Vector3D } b = \text{Vector3D}[i][j-1] - \text{Vector3D}[i][j+1]$
  - $\text{Vector3D}(a,b)$  (-> Kreuzprodukt)
  
- Problem:
  - Jeder Punkt bräuchte für jede angrenzende Fläche eine Normale (4 Flächen grenzen an).
  
- Sieht aber trotzdem gut aus!

# Texturen / Rendern

- Textur Laden
  - `texWasser = new CAMP::Bitmap();`
  - `texWasser->load(„file“);`
- Texturbild definieren
  - `glTexImage2D(GL_TEXTURE_2D, 0, 3, texWasser->getWidth(), texWasser->getHeight(), 0, GL_BGR_EXT, GL_UNSIGNED_BYTE, texWasser->getData());`
- Texturkoordinaten setzen
  - `glTexCoord2f(x, y);` -> legt die Bildkoordinate auf
  - `glVertex3f(x, y, z);` -> den nachfolgenden Punkt

# Collision Detection

Martin Groher

Adopted from NeHe Lesson 30

03 July 2006

chair for computer aided medical procedures

department of computer science | technische universität münchen

# Kollisionserkennung – Strahl und Ebene

- Strahl:  $\mathbf{p} = \mathbf{p}_0 + t \cdot \mathbf{d}$  für  $t \in [0, \infty]$
- Ebene:  $\mathbf{X}_n \circ \mathbf{X} = d$
- Schnittpunkt des Strahls mit der Ebene
  - Zwei Gleichungen:

$$\mathbf{X}_n \circ \mathbf{p} = d \quad \text{oder} \quad (\mathbf{X}_n \circ \mathbf{p}_0) + t \cdot (\mathbf{X}_n \circ \mathbf{d}) = d$$

- Nach t auflösen: 
$$t = \frac{(d - \mathbf{X}_n \circ \mathbf{p}_0)}{(\mathbf{X}_n \circ \mathbf{d})}$$

- d ersetzen: 
$$t = \frac{(\mathbf{X}_n \circ \mathbf{p} - \mathbf{X}_n \circ \mathbf{p}_0)}{(\mathbf{X}_n \circ \mathbf{d})} = \frac{\mathbf{X}_n \circ (\mathbf{p} - \mathbf{p}_0)}{(\mathbf{X}_n \circ \mathbf{d})}$$

- t ist der Abstand eines Punktes „Raystart“ zur Ebene entlang der Achse „Raydirection“.

# Code Snippet

```
TestIntersionPlane(const Plane& plane,const Vector3<double>& position,const
Vector3<double>& direction, double& lamda, Vector3<double>& pNormal)
{
    double DotProduct = dot(direction, plane.normalVal);
    double l2;
    //determine if ray parallele to plane
    if ((DotProduct<ZERO)&&(DotProduct>-ZERO))
        return 0;

    l2=dot(plane.normalVal, plane.positionVal-position)/DotProduct;

    if (l2<-ZERO)
        return 0;

    pNormal=plane.normalVal;
    lamda=l2;
    return 1;
}
```

# Code Snippet

```
rt2= (ballOldPos[i]-m_ballPos[i]).norm();
if (TestIntersionPlane(m_pl1,ballOldPos[i],uveloc,rt,norm))
{
    //Find intersection time
    rt4=rt*RestTime/rt2;
    //if smaller than the one already stored replace and in timestep
    if (rt4<=lamda)
    {
        if (rt4<=RestTime+ZERO)
            if (! ((rt<=ZERO)&&(dot(uveloc,norm)>ZERO)) )
            {
                normal=norm;
                point=ballOldPos[i]+uveloc*rt;
                lamda=rt4;
                BallNr=i;
            }
    }
}
```

# Kollisionserkennung – Kugel-Kugel

- Eine Kugel ist durch Ihren Mittelpunkt  $\mathbf{m}$  und Radius  $r$  definiert
- Zwei Kugel kollidieren, wenn der Abstand zwischen den Mittelpunkten der Kugeln kleiner ist als die Summe beider Radien

$$\|\mathbf{m}_1 - \mathbf{m}_2\|_2 < r_1 + r_2$$

- Komplizierter, wenn man eine komplette diskrete Strecke auf Kollision überprüfen will
  - Ray beider Kugeln berechnen
  - Diskrete Intervalle überprüfen

# Code Snippet

```
glNewList(m_dlist=glGenLists(1), GL_COMPILE);
glBegin(GL_QUADS);
    glRotatef(-45,0,1,0);
    glNormal3f(0,0,1);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-50,-40,0);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(50,-40,0);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(50,40,0);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-50,40,0);
glNormal3f(0,0,-1);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-50,40,0);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(50,40,0);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(50,-40,0);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-50,-40,0);
glNormal3f(1,0,0);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(0,-40,50);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(0,-40,-50);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(0,40,-50);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(0,40,50);
glNormal3f(-1,0,0);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(0,40,50);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(0,40,-50);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(0,-40,-50);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(0,-40,50);
glEnd();
glEndList();
```

# Code Snippet

```
glEnable(GL_BLEND);
glDepthMask(GL_FALSE);
glBindTexture(GL_TEXTURE_2D, m_texture[1]);
for(i=0; i<20; i++) // max no of explosions 20...
{
    if(m_explosionArray[i].alphaVal>=0)
    {
        glPushMatrix();
        m_explosionArray[i].alphaVal-=0.01f;
        m_explosionArray[i].scaleVal+=0.03f;
        glColor4f(1,1,0,m_explosionArray[i].alphaVal);
        glScalef(m_explosionArray[i].scaleVal, m_explosionArray[i].scaleVal,
            m_explosionArray[i].scaleVal);
        glTranslatef(
            (float)m_explosionArray[i].positionVal.getX()/m_explosionArray[i].scaleVal,
            (float)m_explosionArray[i].positionVal.getY()/m_explosionArray[i].scaleVal,
            (float)m_explosionArray[i].positionVal.getZ()/m_explosionArray[i].scaleVal);
        glCallList(m_dlist);
        glPopMatrix();
    }
}
glDepthMask(GL_TRUE);
glDisable(GL_BLEND);
glDisable(GL_TEXTURE_2D);
```