Seminar: Stereoscopic 3D Vision

Stereoskopie in OpenGL am Beispiel von Molekülvisualisierung

Vortrag von:

Thomas Waltemate

Stereoskopie in OpenGL am Beispiel von Molekülvisualisierung

- **Einführung**
- **Mathematischer Hintergrund**
- **OpenGL**
- **Molekülvisualisierung**
- **Fazit**

Einführung

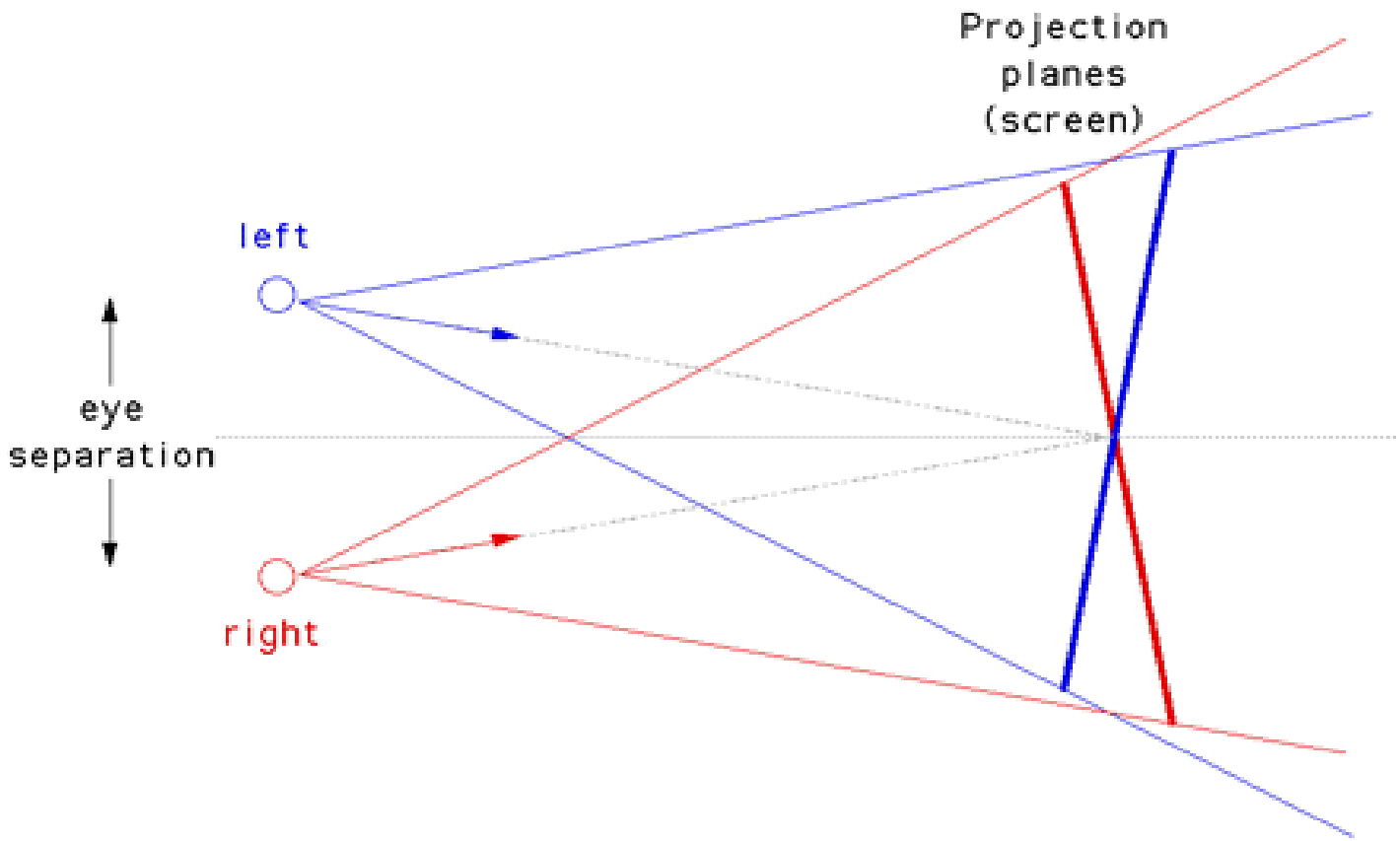
- **Benötigt**
  - Zwei verschiedene Bilder: Linkes + rechtes Auge

- **Zwei grundsätzliche Ansätze**
  - Toe-in
  - Off-axis

Einführung

Toe-in

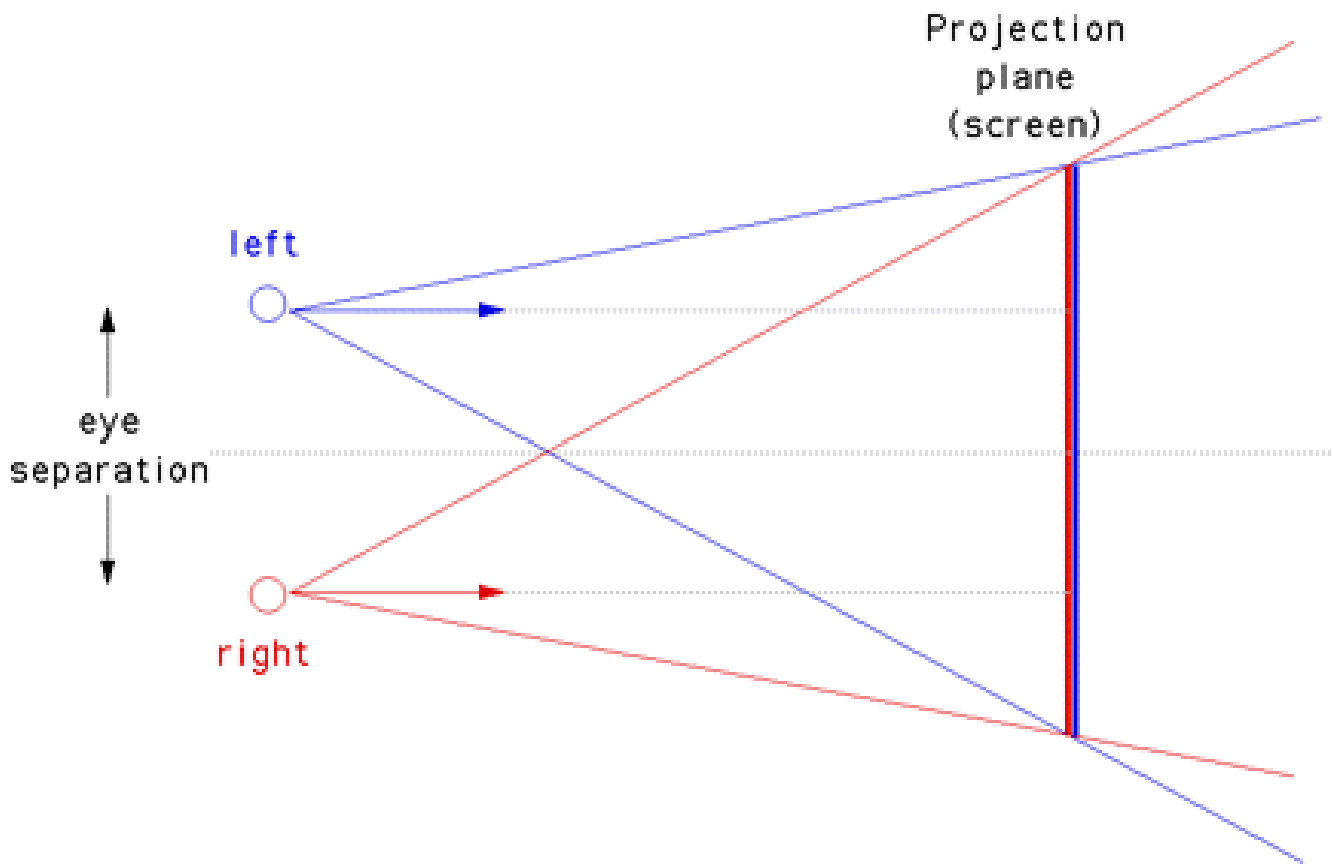
- **Zwei Bilder, die sich kreuzen**



Einführung

Off-axis

- **Zwei Bilder, die eine Ebene bilden**



Einführung

Richtiger Ansatz

- **Richtig: Off-axis**
  - Die Anzeige selbst liegt auch nur in einer Ebene
  - Sorgt dafür, dass es kein bzw. weniger Unwohlsein und Kopfschmerzen gibt
- **Falsch: Toe-in**
  - Sinnvoll, wenn Frustum Berechnung nicht verfügbar

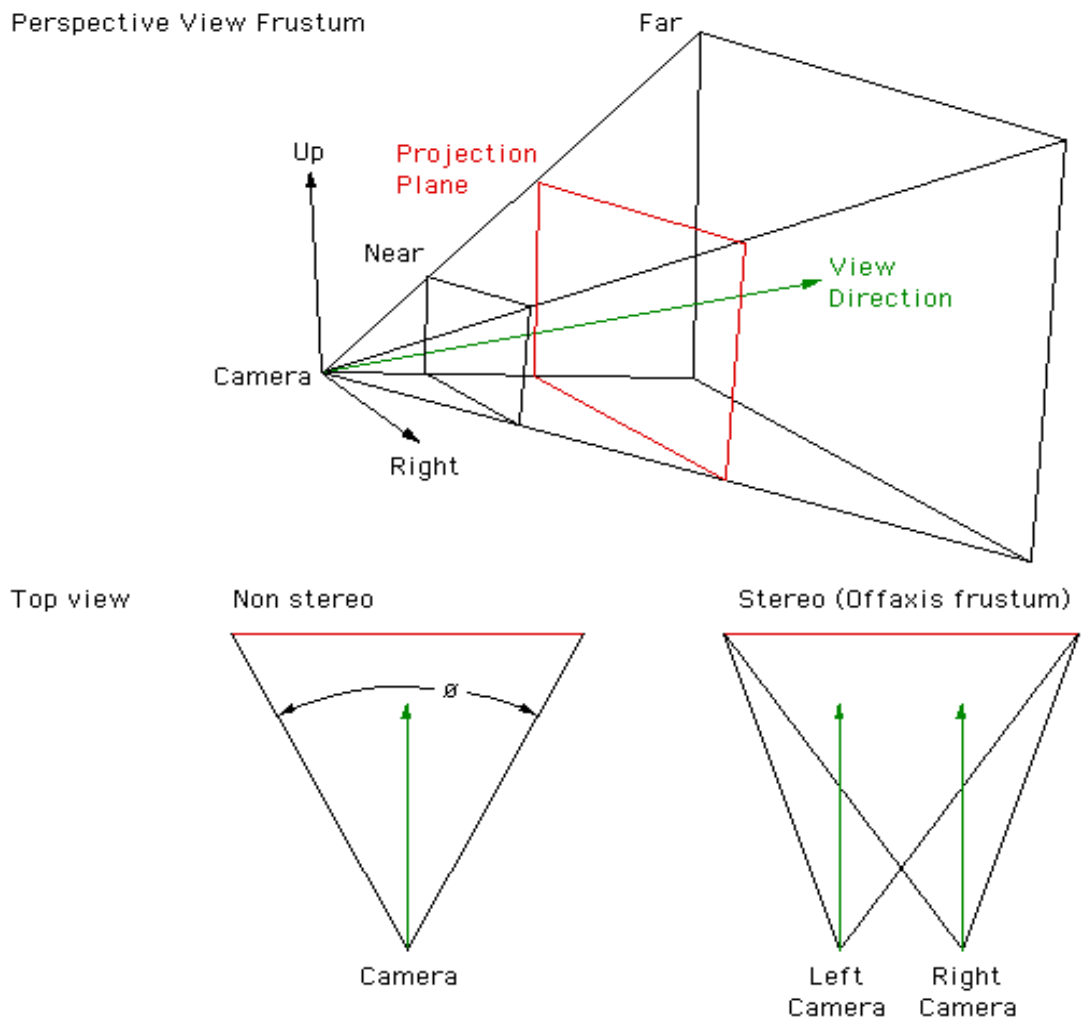
Einführung

Frustum

- **Was ist ein Frustum?**
  - Wikipedia: „Frustum ist in der Computergrafik die mathematische Abbildung eines 3D-Universums auf den Bildschirm“
  - Also im Wesentlichen der Bereich, der gerendert wird bzw. auf den der Benutzer blickt
  - In unserem Fall: Zwei Frusta/Frustums
  - Das, was hauptsächlich „berechnet“ werden muss

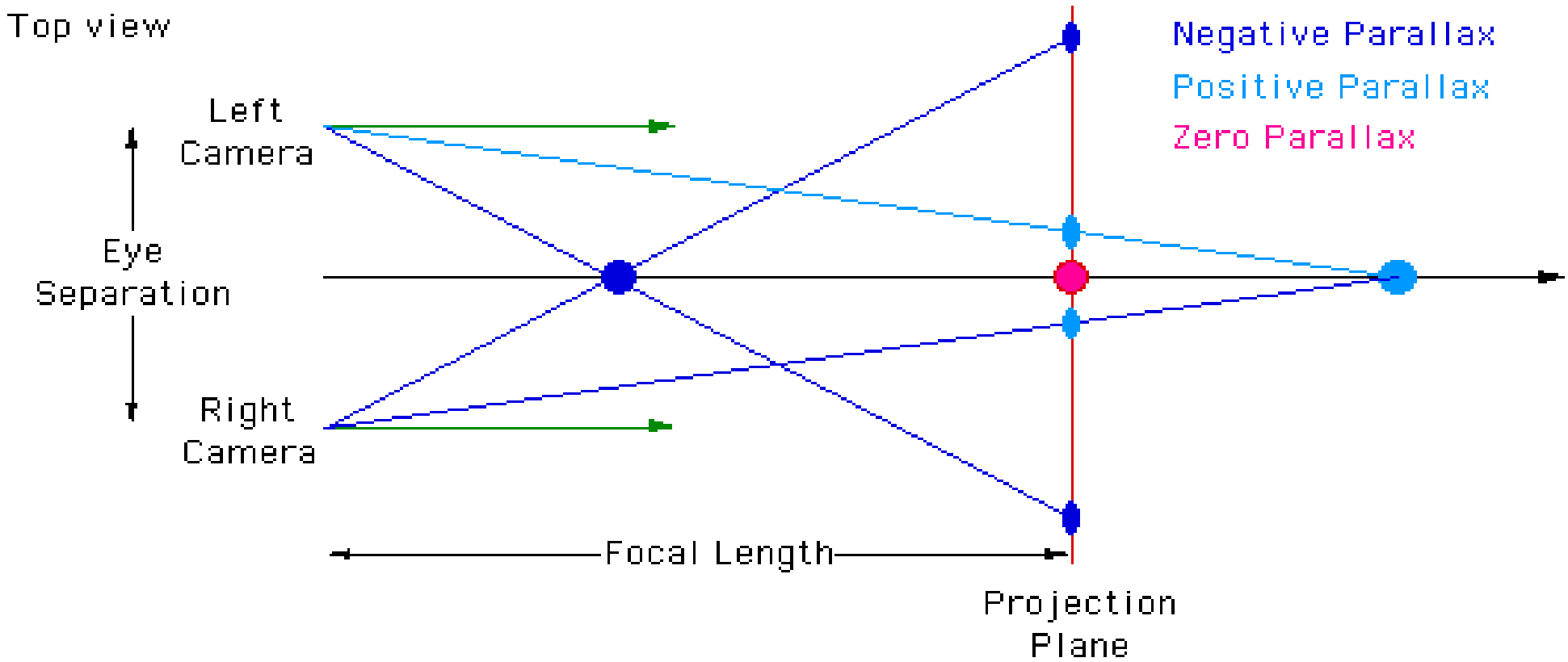
Einführung

Frustum



Einführung

Frustum



- **„Positive Parallax“ einfacher zu sehen**

Mathematischer Hintergrund

Grundlagen

- **Zwei grundlegende Schritte nötig**
  - Die Szene/Kamera transformieren
  - Das Frustum berechnen

Mathematischer Hintergrund

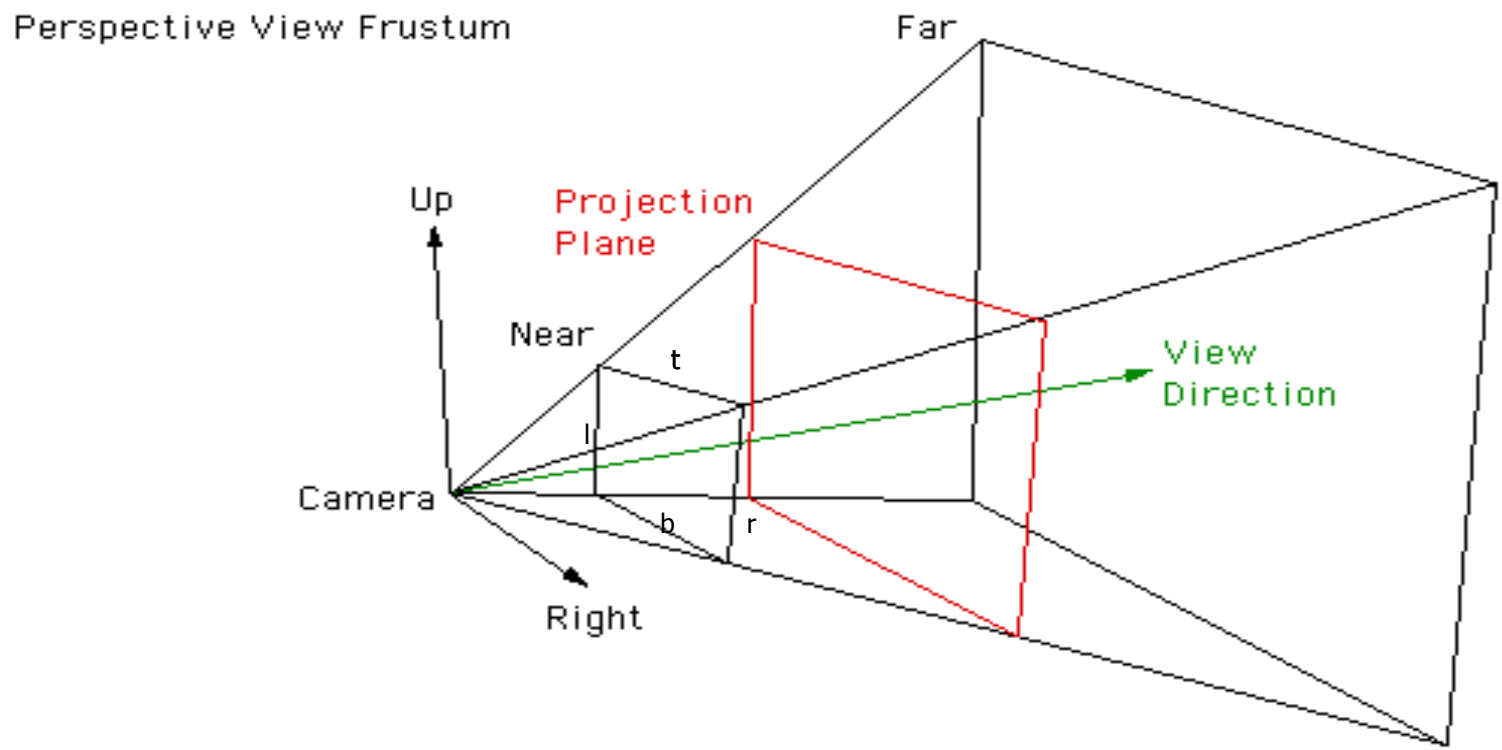
Szene/Kamera transformieren

- **Szene transformieren**
  - Einfach: glTranslate(+/-(0.5*eye_separation), 0.0, 0.0)
- **Kamera transformieren**
  - Etwas schwieriger: gluLookAt(...)
- **Natürlich ist nur eines von Beiden nötig!**

Mathematischer Hintergrund

Frustum berechnen

- **Benötigte Werte für Frustum:**
  - Left l
  - Right r
  - Bottom b
  - Top t
  - Near near
  - Far far



Mathematischer Hintergrund

Frustum berechnen

- **Berechnung (Top und Bottom)**

$$wd2 = near * \tan\left(\left(\frac{\pi}{180}\right) * \left(\frac{FOVy}{2}\right)\right)$$

wd2: „Öffnung / 2"

$$b = -wd2$$

FOVy: Field Of View y
„Öffnungswinkel in y"

$$t = wd2$$

- **Konstanten**

Mathematischer Hintergrund

Frustum berechnen

- **Berechnung (Left und Right)**
  - Linkes Auge

$$l = b * \left(\frac{width}{height}\right) + 0.5 * eyeseparation * \left(\frac{near}{focaldistance}\right)$$

$$r = t * \left(\frac{width}{height}\right) + 0.5 * eyeseparation * \left(\frac{near}{focaldistance}\right)$$

  - Rechtes Auge

$$l = b * \left(\frac{width}{height}\right) - 0.5 * eyeseparation * \left(\frac{near}{focaldistance}\right)$$

$$r = t * \left(\frac{width}{height}\right) - 0.5 * eyeseparation * \left(\frac{near}{focaldistance}\right)$$

Mathematischer Hintergrund

Fertig?!

- **Sind wir damit fertig?**
  - Ja, dank OpenGL
  - Denn...

```
void glFrustum(
        GLdouble  left,
        GLdouble  right,
        GLdouble  bottom,
        GLdouble  top,
        GLdouble  nearVal,
        GLdouble  farVal);
```

OpenGL

Grundlagen

- **Was ist OpenGL?**
  - Open Graphics Library
- **Wie arbeitet OpenGL?**
  - State Machine: Status bleibt so lange bis Status wieder aktiv geändert wird!
- **Framework**
  - Hier: GLUT - OpenGL Utility Toolkit

OpenGL

Rendering

- **Ablauf:**
  1. Berechne Frustum für das linke Auge
  2. Transformiere Szene/Kamera um die Hälfte des Augenabstands nach rechts/links
  3. Rendere die Szene
  4. Berechne Frustum für das rechte Auge
  5. Transformiere Szene/Kamera um die Hälfte des Augenabstands nach links/rechts
  6. Rendere die Szene
- (Code zeigen...)

OpenGL

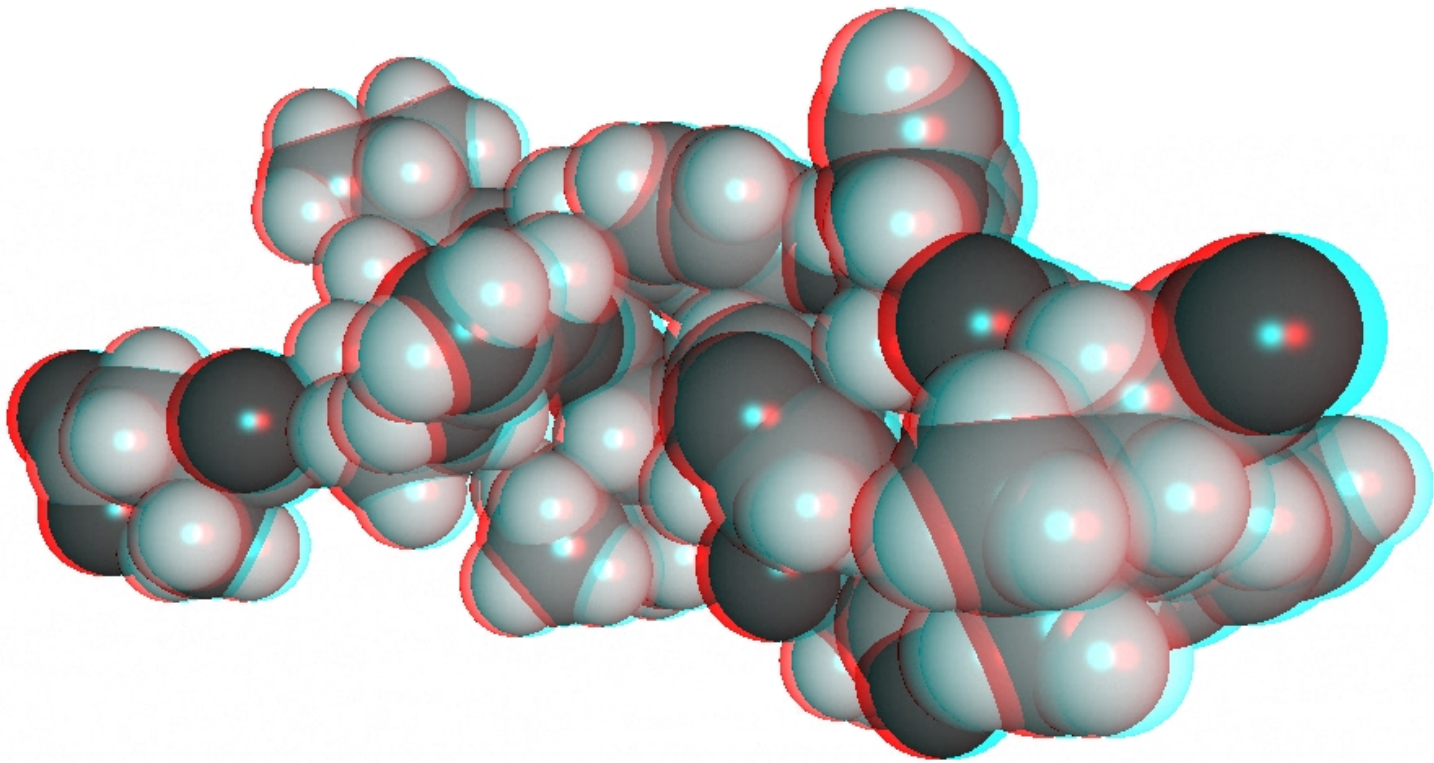
Rendering: Rot Cyan

- **Ablauf wie vorher**
- **Einziger Unterschied**
  - Vor einzelnen Renderdurchgängen Farbfilter setzen
  - glColorMask(bool red, bool green, bool blue, bool alpha);
- **Linkes Auge**
  - glColorMask(GL_TRUE, GL_FALSE, GL_FALSE, GL_FALSE);
- **Rechtes Auge**
  - glColorMask(GL_FALSE, GL_TRUE, GL_TRUE, GL_FALSE);
- (Code und Ergebnis zeigen...)

OpenGL

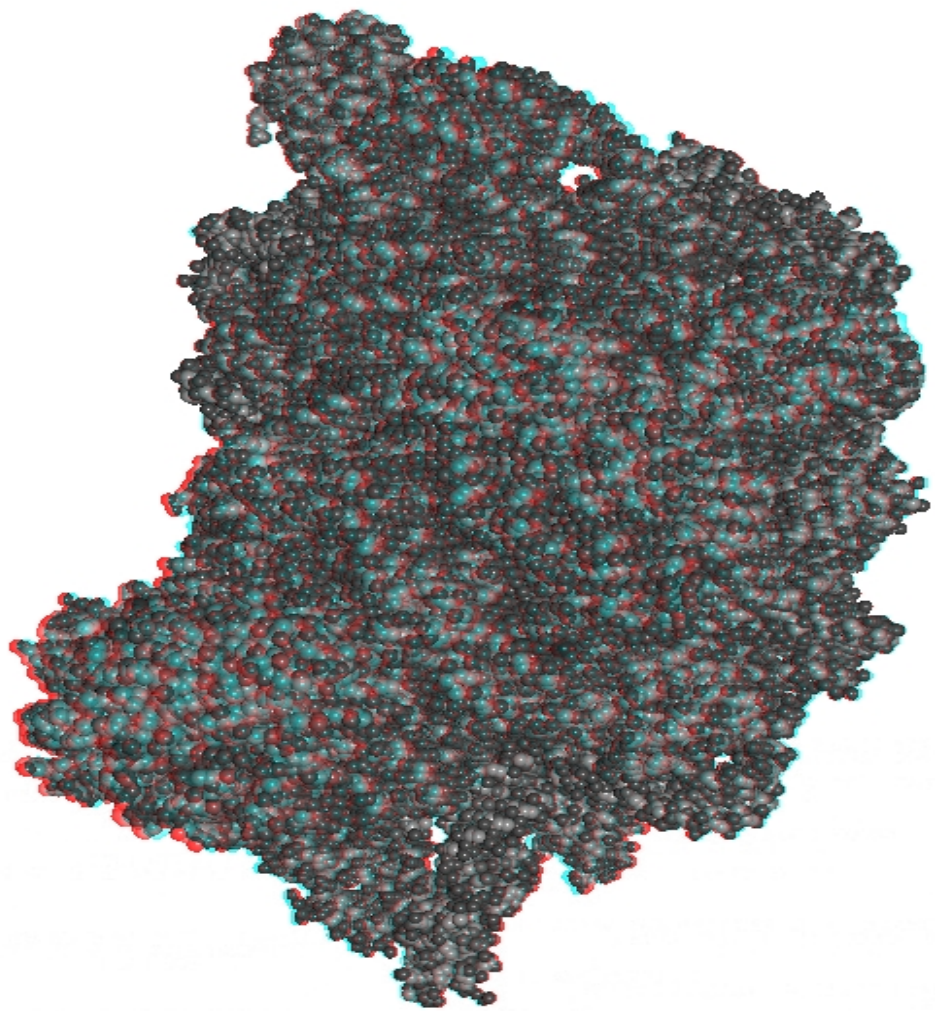
Rendering: Rot Cyan

Interessanter „Vorteil“ der Rot-Cyan-Darstellung:



OpenGL

Rendering: Rot Cyan



OpenGL

Rendering: mehrere DrawBuffer

- **Ablauf wieder wie vorher**
- **Unterschied**
  - Aktiviere Stereo (in GLUT)
    - glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE | GLUT_STEREO);
  - Schreibe das linke Bild in linken Drawbuffer...
    - glDrawBuffer(GL_BACK_LEFT);
  - ...und das rechte in den rechten Drawbuffer
    - glDrawBuffer(GL_BACK_RIGHT);
- (Code und Ergebnis zeigen...)

Molekülvisualisierung

Warum Moleküle?

- **Bei (Bio-)Molekülen ist es oft besonders wichtig einzelne Bereiche unterscheiden zu können**
- **Bei 2D Darstellung nur bedingt bzw. oft nur bei Bewegung möglich**
- **(Aufwändige Schattenberechnung würde das Problem ebenfalls zum Teil lösen)**

Moleküldarstellung

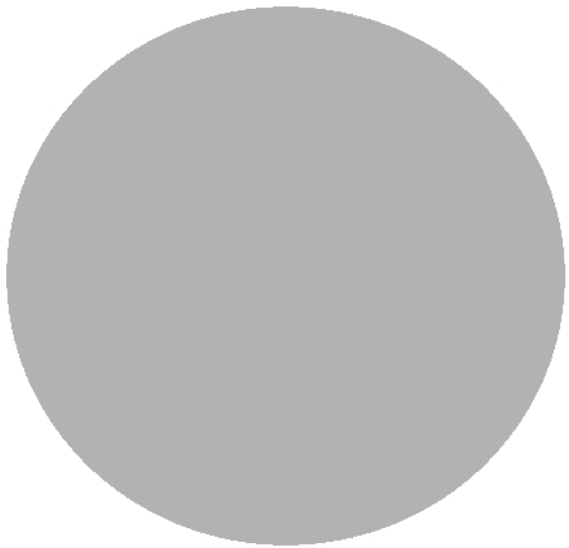
Effizienz

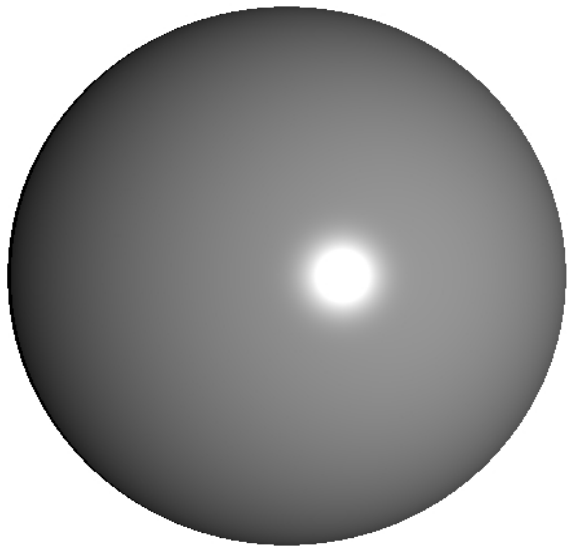
- **Effiziente Rendermethoden nötig**
  - Molekül muss zwei mal gerendert werden für 3D-Darstellung
    ==> optimalerweise min. 24 fps bzw. sogar 120 fps (Shutterbrille)
- **Hier: Effizienz durch effiziente Shaderprogramme**
  - (Ray Casting of Quadratic Surfaces)
- **Einfacher Ansatz über Dreiecke und Vertices ist viel zu langsam!**

Moleküldarstellung

Ray Casting of Quadratic Surfaces

- **Basiert auf Shadern, die Nullstellenfunktionen (Quadrics) auswerten**





Fazit

- **Off-axis angenehmer bzw. „richtiger“ als Toe-in**
- **Schwieriger Teil der Rechnung: Das Frustum**
  - In OpenGL bei bekannter „Formel“ relativ einfach
- **Moleküldarstellung**
  - Wichtiges Anwendungsgebiet der Stereoskopie

Vielen Dank für die Aufmerksamkeit!

Fragen?!

Quellen

- **Calculating Stereo Pairs von Paul Bourke**
- **http://www.opengl.org/**
- **Bachelorarbeit: GPU-Based Molecule Rendering**