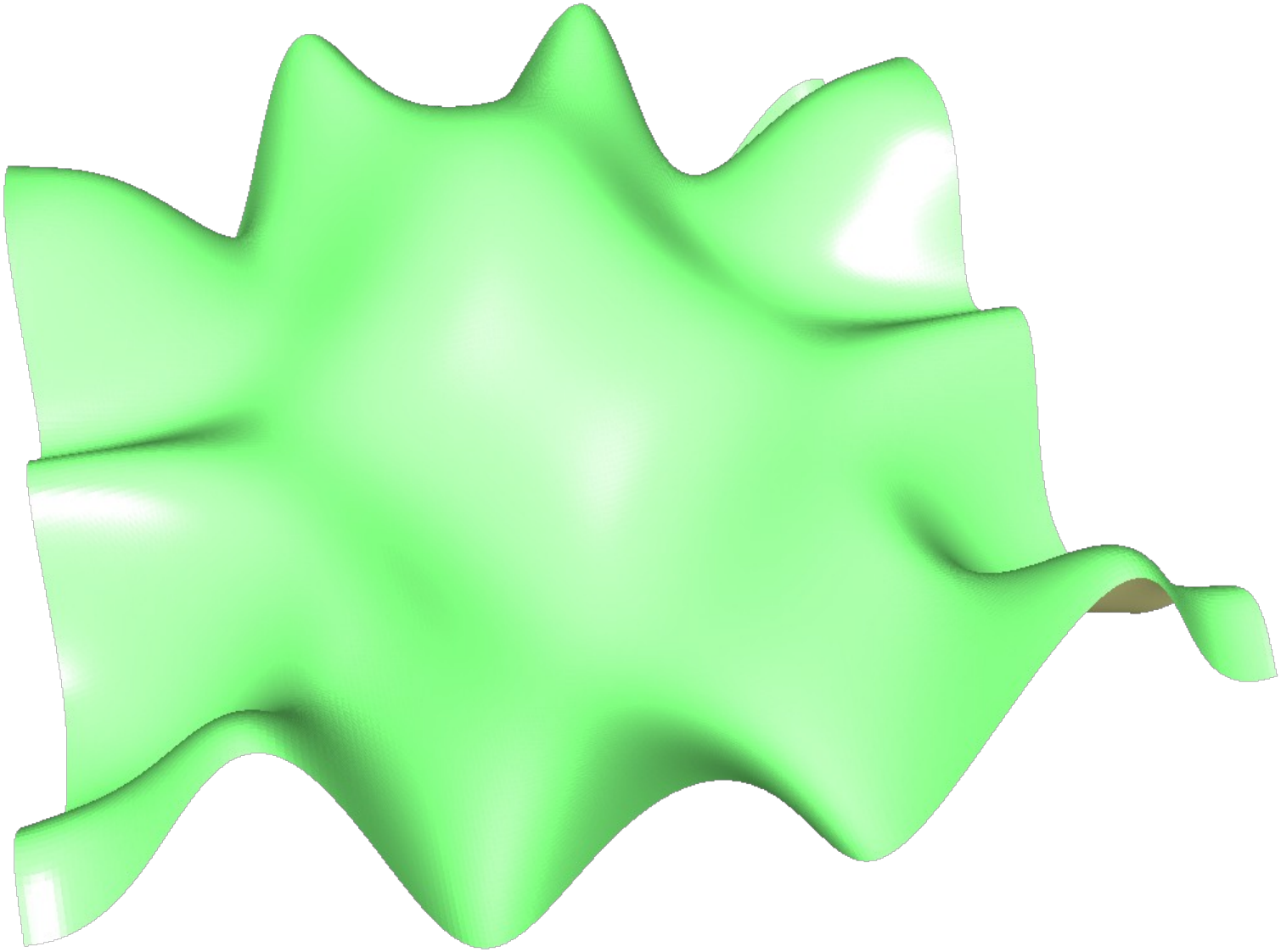
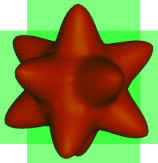
# OpenGL II

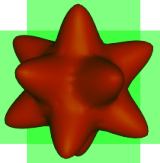




# Gliederung

- Blending
- Licht
- Anwendungsbeispiel
- Optimierung
- OpenGL „heute und morgen“

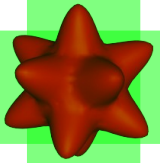




# Blending

- Entscheidung, was passiert, wenn sich zwei Objekte „überlappen“
- Einstellung in OpenGL über glBlendFunc(source, destination)
- $F_N = P * S + F_V * D$
  - $F_N$ – Framebuffer nach der Verrechnung
  - $F_V$ – Framebuffer vor der Berechnung
  - P – der „neue“ Pixel, der mit FV interagiert
  - S, D – die übergebenen Werte

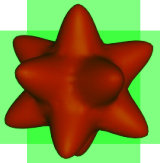




# Blending

- Mögliche Werte in OpenGL für source und destination (Auszug):
  - GL_ZERO (ignorieren, Multiplikation mit 0)
  - GL_ONE (alles nehmen, Multipl. Mit 1)
  - GL_SRC_ALPHA (Alphawert der Quelle)
  - GL_ONE_MINUS_SRC_ALPHA (1 - α d. Q.)
  - GL_DST_ALPHA (Alphawert des Ziels)
  - GL_ONE_MINUS_SRC_ALPHA
  - ...





# Multitexturing

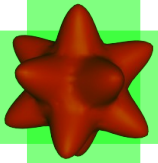
- mehrere Texturen, die gleichzeitig auf eine Fläche gemacht werden

  => mehrere UV Koordinaten pro Vertex

  => Über glTexEnv wird angegeben, wie die verschiedenen Texturen „verrechnet“ werden (ähnlich zu Alpha Blending)

- Vorteile:
  - u.U. weniger (langsame) Texturwechsel
  - Weniger Daten müssen über den BUS

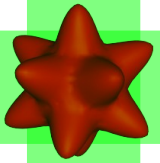




# Licht

- Vorteile:
  - Relativ schnell
  - Auf älteren Rechnern verfügbar
  - Für simple Szenen ausreichend
- Nachteil
  - Sehr einfach
  - Ab vielen Lichtquellen „Lichtbrei“
  - Keine Schatten
  - Keine Reflektion

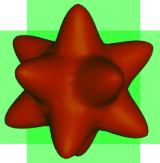




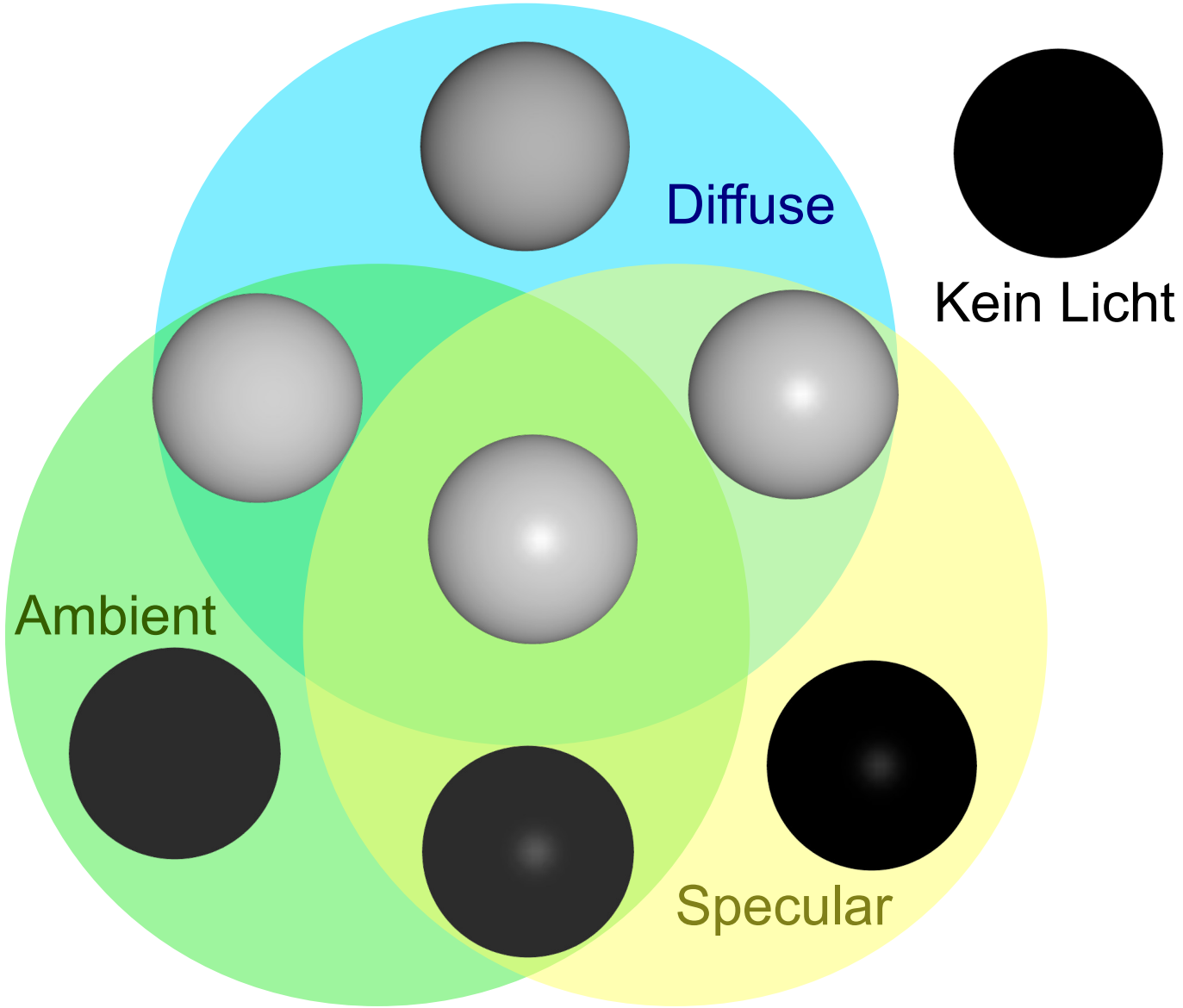
# Licht

- Eigenschaften von OpenGL-Licht:
  - Ambient: Licht aus allen Richtungen
  - Diffuse: Licht aus einer best. Richtung
  - Specular: Glanz aus einer best. Richtung
  - Farbe für ambient, diffus & spekular
  - Position
  - Licht kann in alle Richtungen oder wie ein Spot in einer Richtung leuchten
  - mehrere Lichter sind möglich (min. 8)





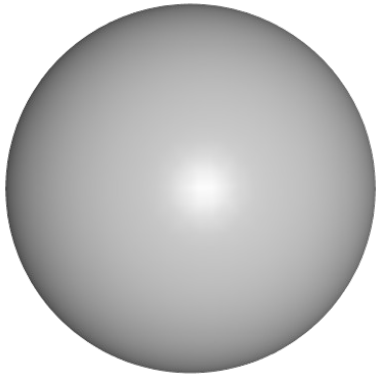
# Licht - Lichtarten im Vergleich

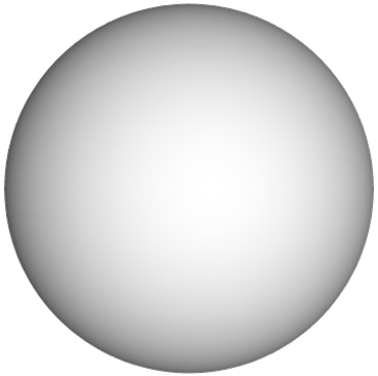


# Licht - Materialeigenschaften

- Wie bei Lichtwerten
- Geben jedoch den Reflexionsgrad wieder
- Zusätzlich noch Parameter für
  - Shininess: Wie stark ist der Glanz?
  - Emission: Eigenleuchten
    - Das Eigenleuchten hat jedoch keinen Einfluss auf die Lichtberechnung der anderen Objekte

Niedriger Shininesswert

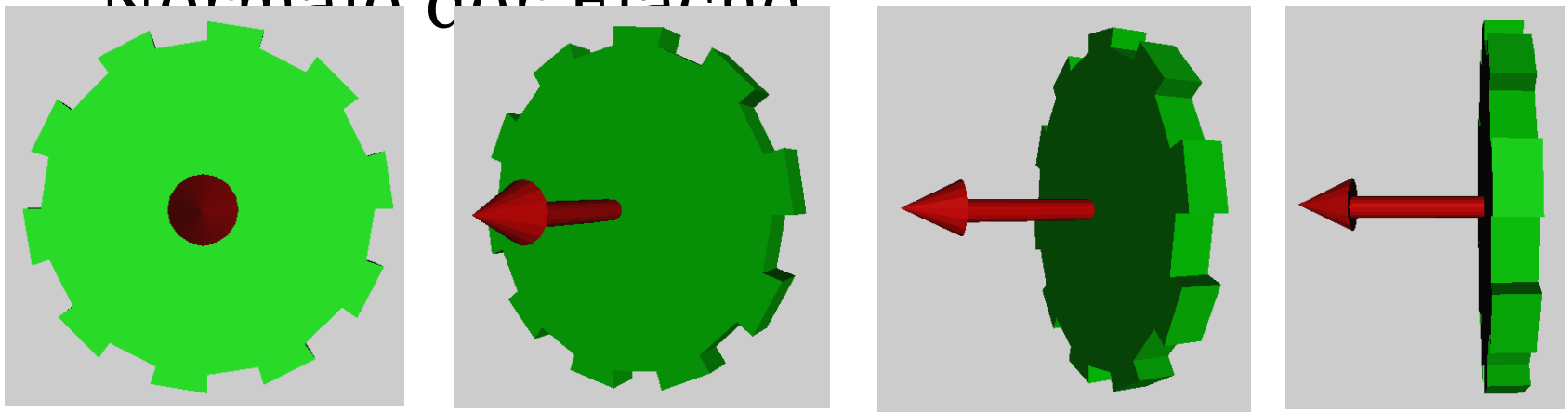




Hoher Shininesswert

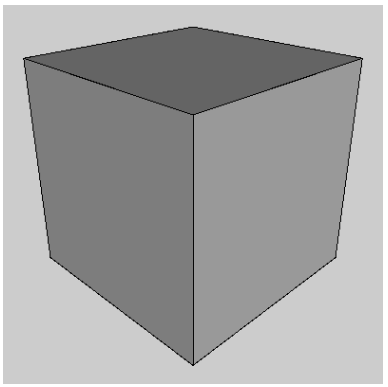
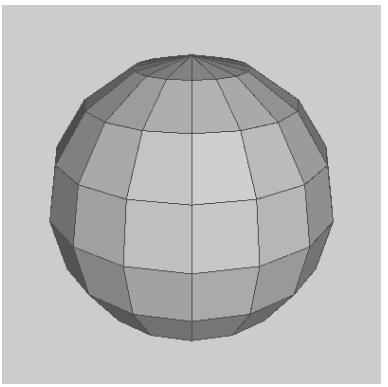
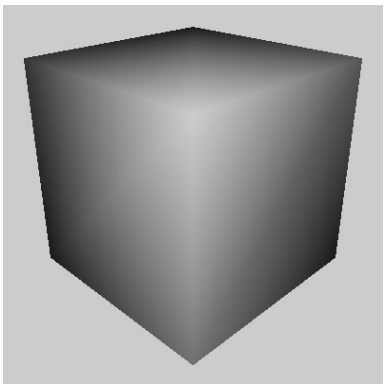
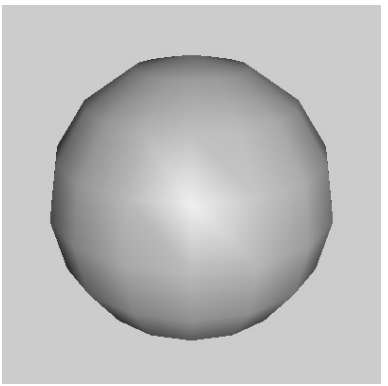
# Licht - Bedeutung der Normalen

- Normale für Lichtberechnung essentiell
- Je mehr Normale d. Fläche bzw. Punkts zur Lichtquelle zeigt, desto mehr „wirkt“ der diffuse Lichtteil bei der Farbberechnung.
- Ambientes Licht ist unabhängig von der Normale der Fläche

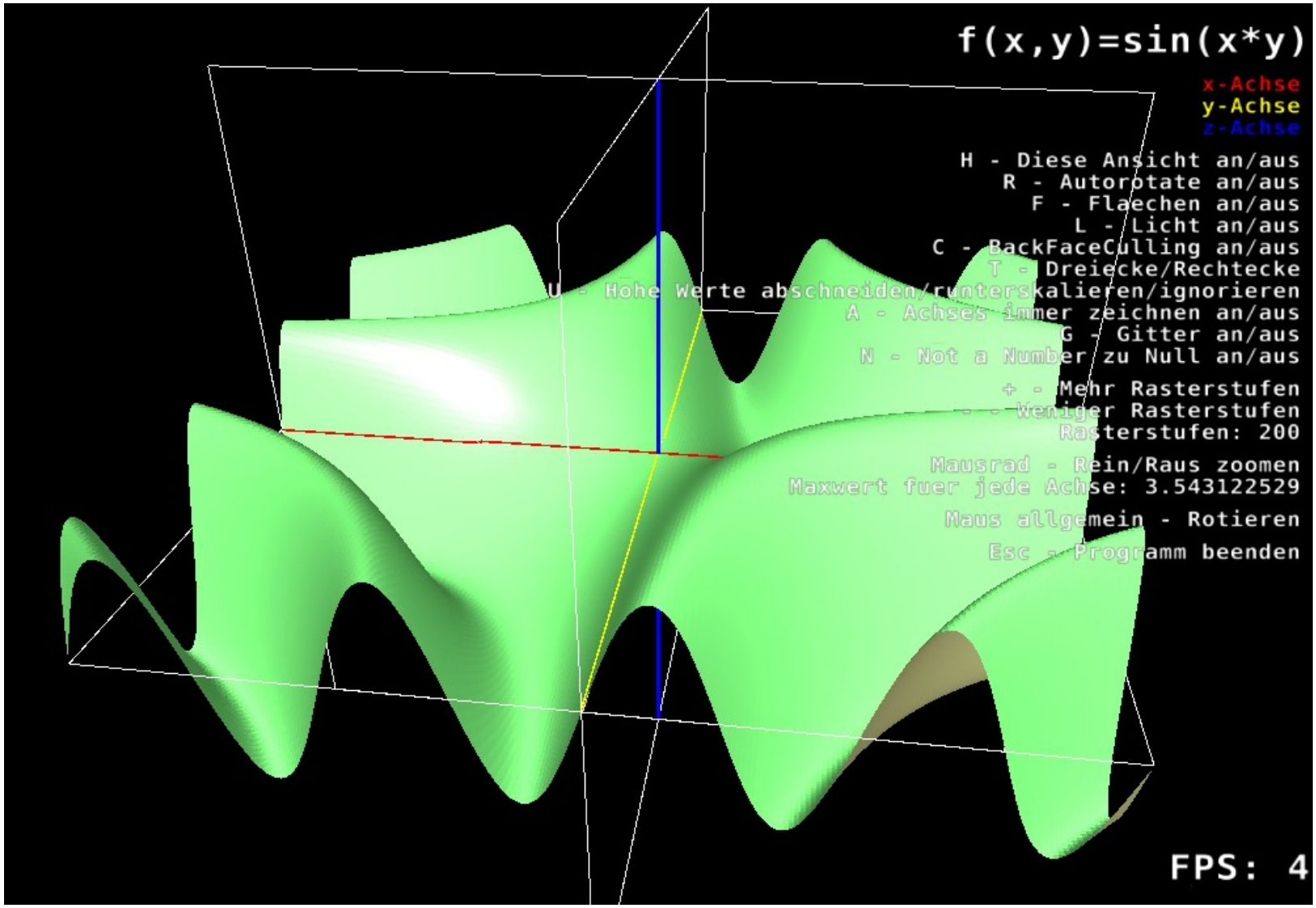


# Licht – Bedeutung der Normalen

- Unterschied zwischen der Berechnung einer Normalen pro Fläche oder Punkt

| | Hohe Kantenwinkel | Niedrige Kantenwinkel |
|---|---|---|
| Normale pro Fläche |  |  |
| Normale pro Punkt |  |  |

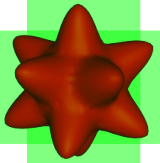
# Anwendungsbeispiel



# Optimierungen - Displaylisten

- Problem:
  - In jedem Frame werden die zu zeichnenden Daten neu über den BUS übertragen mit glBegin, glVertex, glTexCoord, etc.
- Lösung:
  - Einmaliges „Aufnehmen“ Sequenz von OpenGL befehlen und speichern im Grafikspeicher
  - Nur noch ein Befehl zum Zeichnen der Displayliste muss über den BUS

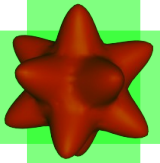




# Optimierungen - Vorsortierung

- Problem:
  - OpenGL zeichnet Primitive in den Framebuffer, welche später überzeichnet werden
- Lösung:
  - Vorsortierung nach Abstand zum Bildschirm, je weiter dran, desto eher
- Achtung:
  - Transparente Objekte müssen zuletzt und in anderer Reihenfolge gezeichnet werden

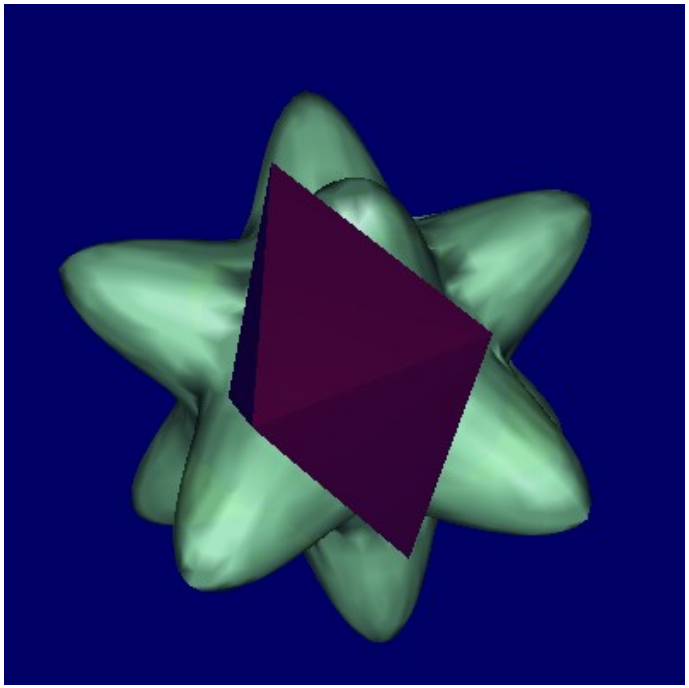




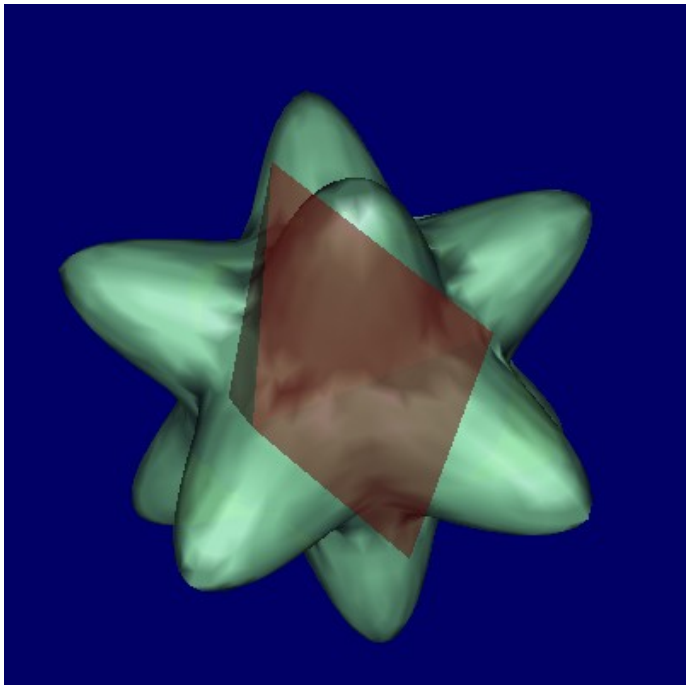
# Optimierungen - Vorsortierung

- Veranschaulichung einer falschen Vorsortierung bei transparenten Objekten

Falsch



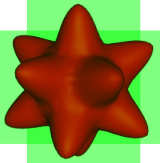
Richtig



# Optimierungen - Frustumculling

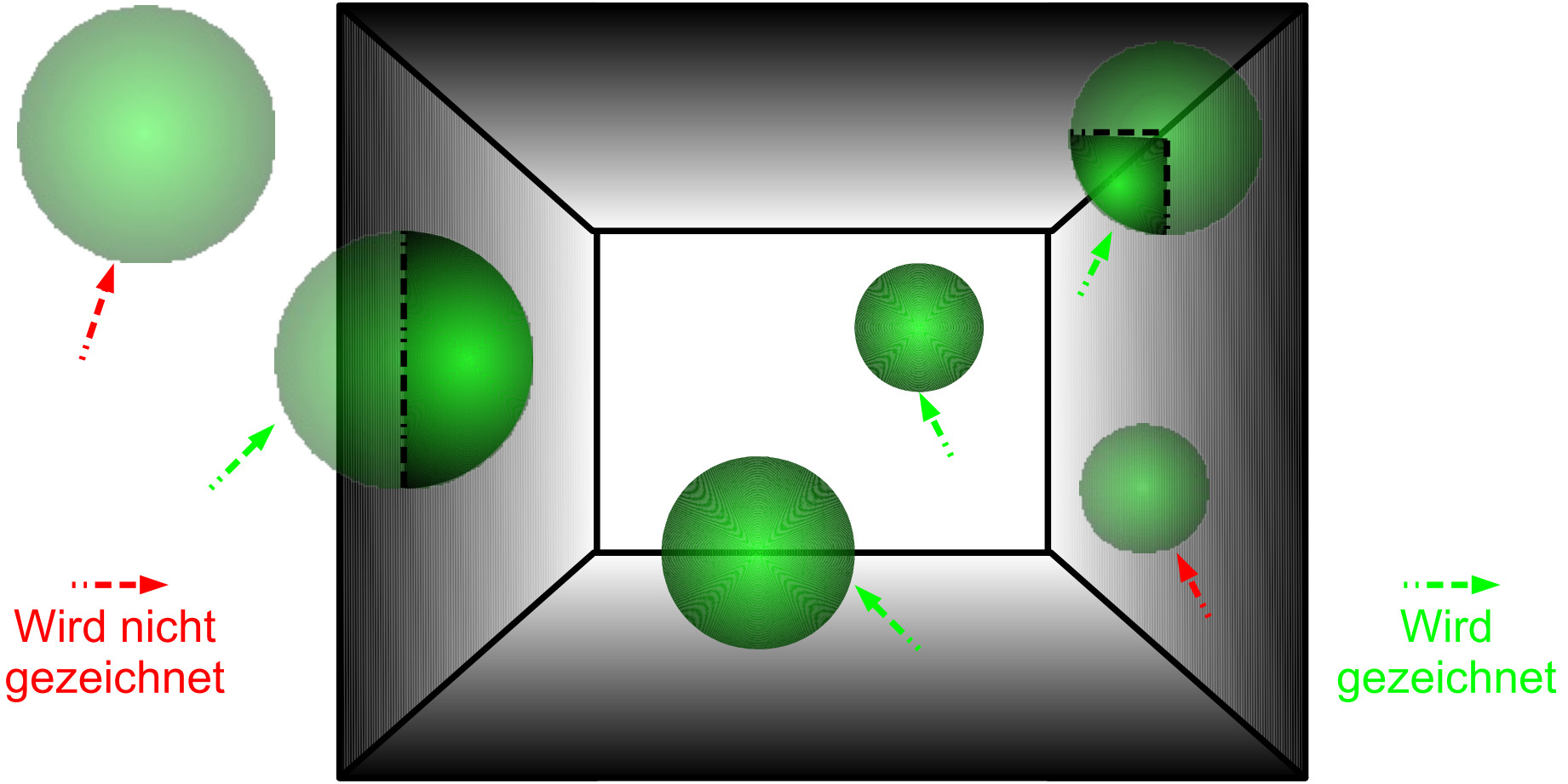
- Problem:
  - Man sagt OpenGL, es soll Objekte an Stellen außerhalb des Bildschirm zeichnen
- Lösung:
  - Vorberechnung, ob Objekt überhaupt im Sichtfeld liegt
  - Sichtbarer Bereich ist ein Pyramidenstumpf
  - Die umhüllende Kugel zu einem Objekt muss in diesem Pyramidenstumpf liegen





# Optimierungen - Frustumculling

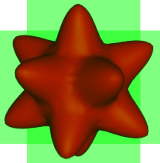
- Veranschaulichung von Frustumculling



# OpenGL „heute und morgen“

- Das „moderne“ OpenGL
  - Weiterentwicklung statt vom OpenGL ARB von der Khronos Gruppe
  - Weg von der fixen Renderpipeline
  - Hin zu einer flexiblen über Shader veränderbaren Pipeline
  - Seit OpenGL 3.0 glBegin(), glEnd() und einiges anderes „deprecated“
  - OpenGL 4.0 zieht funktional mit Direct3D 11 gleich

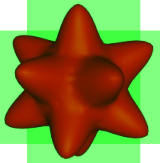




# OpenGL „heute und morgen“

- Die Bedeutung von OpenGL
  - Wenig benutzt bei kommerziellen PC-Spielen, aber dafür bei
    - Plattformunabhängigen 3D-Programmen/Spielen
      - z.B. Blender, OpenArena, OpenParty, Compositingfähige Windowmanager, Beschleunigung z.B. von OpenOffice
    - Als OpenGL ES in Embedded Systemen
      - z.B. Konsolen oder mobile Geräte wie Smartphones
    - Workstations
    - WebGL





# Quellen

- http://wiki.delphigl.com
- http://www.libsdl.org
- Dieter Orlamünder und Wilfried Mascolus „Computergrafik und OpenGL“, Fachbuchverlag Leipzig, 1. Auflage 2004