Grundlagen der Spieleprogrammierung

Teil I: 3D-Graphik

Kapitel 3: Das Ideal - Photorealistisch

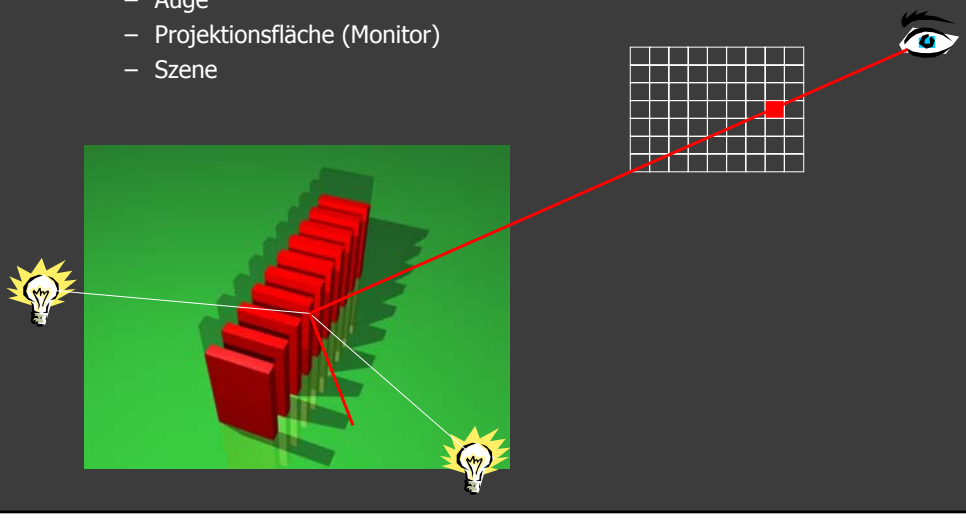
Peter Sturm
Universität Trier

Outline

1. Übersicht und Motivation
2. Mathematische Grundlagen
3. Das Ideal: Photorealistisch (Raytracing, Radiosity)
4. Die Realität: DirectX und OpenGL (Übersicht)
5. Schritt 1: Drahtgitter
6. Schritt 2: Texturen
7. Schritt 4: Licht, Filter, etc.
8. Schritt 5: Fortgeschrittene Techniken (Vertex-, Pixel-Shader, …)
9. 3D-Hardware
10. 3D-Engines im Überblick, Cg von nvidia
11. Spielekonsolen
12. Zusammenfassung und Ausblick

Grundidee: Raytracing

- Direkte Umsetzung des Sehens
  - Auge
  - Projektionsfläche (Monitor)
  - Szene



Raytracer

- Beschreibung der darzustellenden 3D-Szene
  - CSG: Constructive Solid Geometry
- Beschreibung der Materialien
  - Farbe
  - Reflektionseigenschaften
  - Refraktionseigenschaften
  - Oberflächenstruktur
  - Extras: Rauhheit, …
- Lichtquellen
- Kameraposition

Verwendete Software

- Povray (Persistence of Vision, www.povray.org)
  - Binaries und Sourcen für Windows, Linux, …
  - Aktuell Version 3.5
    - 8 MB (Windows)
    - 7 MB (Linux)
- 3D-Modeler für Povray verfügbar
  - teilweise kommerziell
- Syntax der Online-Hilfe entnommen

CSG

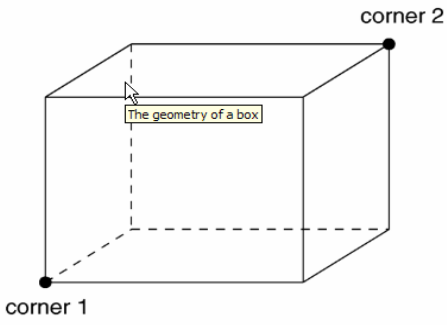
- Volumetrische Grundobjekte
  - Quader, Kugel, Zylinder, …
- Mengentheoretische Konstruktionsoperationen
  - Vereinigung
  - Schnitt
  - Differenz

Finite Solid Primitives

- Gängige Elemente
  - Box
  - Cone
  - Cylinder
  - Prism
  - Sphere
  - Torus
  - Text
- Exoten
  - Blob
  - Height Field
  - Julia Fractal
  - Lathe
  - Superquadric Ellipsoid
  - Surface of Revolution

Box

```
BOX:
    box
    {
        <Corner_1>, <Corner_2>
        [OBJECT_MODIFIERS...]
    }
```

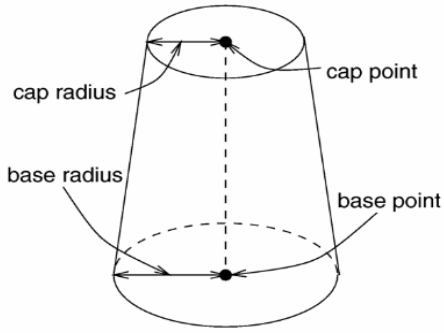


The geometry of a box

Where *<Corner_1>* and *<Corner_2>* are vectors defining the x, y, z coordinates of the opposite corners of the box.

Cone

```
CONE:
    cone
    {
        <Base_Point>, Base_Radius, <Cap_Point>, Cap_Radius
        [ open ][OBJECT_MODIFIERS...]
    }
```

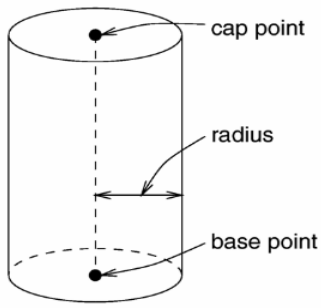


The geometry of a cone

Where *<Base_Point>* and *< Cap_Point>* are vectors defining the x, y, z coordinates of the center of the cone's base and cap and *Base_Radius* and *Cap_Radius* are float values for the corresponding radii.

Cylinder

```
CYLINDER:
    cylinder
    {
        <Base_Point>, <Cap_Point>, Radius
        [ open ][OBJECT_MODIFIERS...]
    }
```

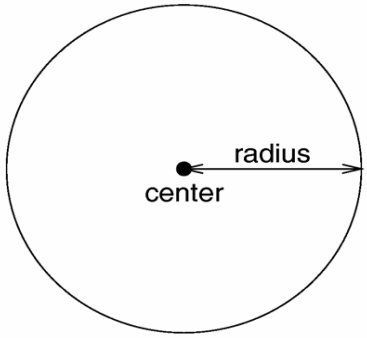


The geometry of a cylinder

Where *<Base_Point>* and *<Cap_Point>* are vectors defining the x, y, z coordinates of the cylinder's base and cap and *Radius* is a float value for the radius.

Sphere

```
SPHERE:
    sphere
    {
        <Center>, Radius
        [OBJECT_MODIFIERS...]
    }
```



The geometry of a sphere

Where `<Center>` is a vector specifying the x, y, z coordinates of the center of the sphere and `Radius` is a float value specifying the radius. Spheres may be scaled unevenly giving an ellipsoid shape.

Superquadric Ellipsoid

The `superellipsoid` object creates a shape known as a *superquadric ellipsoid* object. It is an extension of the quadric ellipsoid. It can be used to create boxes and cylinders with round edges and other interesting shapes. Mathematically it is given by the equation:

$$f(x,y,z)=\left(|x|^{\left(\frac{2}{e}\right)}+|y|^{\left(\frac{2}{e}\right)}\right)^{\left(\frac{e}{n}\right)}+|z|^{\left(\frac{2}{n}\right)}-1=0$$

The values of `e` and `n`, called the *east-west* and *north-south* exponent, determine the shape of the superquadric ellipsoid. Both have to be greater than zero. The sphere is given by *e = 1* and *n = 1*.

The syntax of the superquadric ellipsoid is:

```
SUPERELLIPSOID:
    superellipsoid
    {
        <Value_E, Value_N>
        [OBJECT_MODIFIERS...]
    }
```

The 2-D vector specifies the `e` and `n` values in the equation above. The object sits at the origin and occupies a space about the size of a `box{<-1,-1,-1>,<1,1,1>}`.

Two useful objects are the rounded box and the rounded cylinder. These are declared in the following way.

```
#declare Rounded_Box = superellipsoid { <Round, Round> }
#declare Rounded_Cylinder = superellipsoid { <1, Round> }
```

Infinite Solid Primitives: Plane

The `plane` primitive is a simple way to define an infinite flat surface. The plane is not a thin boundary or can be compared to a sheet of paper. A plane is a solid object of infinite size that divides POV-space in two parts, inside and outside the plane. The plane is specified as follows:

```
PLANE:
    plane
    {
        <Normal>, Distance
        [OBJECT_MODIFIERS...]
    }
```

The *<Normal>* vector defines the surface normal of the plane. A surface normal is a vector which points up from the surface at a 90 degree angle. This is followed by a float value that gives the distance along the normal that the plane is from the origin (that is only true if the normal vector has unit length; see below). For example:

```
plane { <0, 1, 0>, 4 }
```

This is a plane where straight up is defined in the positive y-direction. The plane is 4 units in that direction away from the origin. Because most planes are defined with surface normals in the direction of an axis you will often see planes defined using the `x`, `y` or `z` built-in vector identifiers. The example above could be specified as:

```
plane { y, 4 }
```

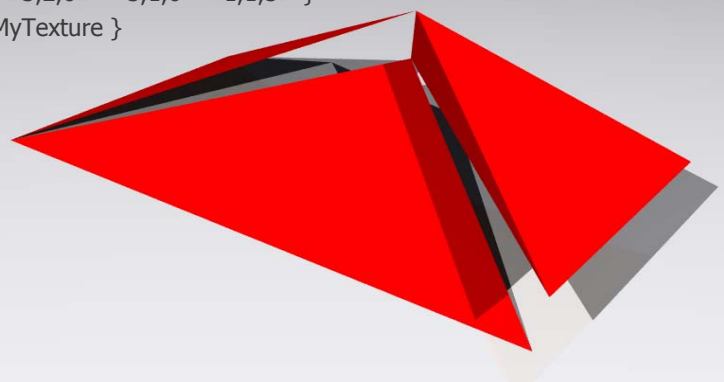
Mesh

- Objektbeschreibungen auf der Basis von Dreiecken

```
MESH:
    mesh
    {
        MESH_TRIANGLE...
        [MESH_MODIFIER...]
    }
MESH_TRIANGLE:
    triangle
    {
        <Corner_1>, <Corner_2>, <Corner_3>
        [uv_vectors <uv_Corner_1>, <uv_Corner_2>, <uv_Corner_3>]
        [MESH_TEXTURE]
    } |
    smooth_triangle
    {
        <Corner_1>, <Normal_1>,
        <Corner_2>, <Normal_2>,
        <Corner_3>, <Normal_3>
        [uv_vectors <uv_Corner_1>, <uv_Corner_2>, <uv_Corner_3>]
        [MESH_TEXTURE]
    }
MESH_TEXTURE:
    texture { TEXTURE_IDENTIFIER }
MESH_MODIFIER:
    inside_vector <direction> | hierarchy [ Boolean ] |
    OBJECT_MODIFIER
```

Mesh: Beispiel

```
mesh {
        triangle { <-3,2,0> <-3,1,5> <1,1,5> }
        triangle { <-3,2,0> <-2,0,-4> <3,0,-4> }
        triangle { <-3,1,0> <1,0,5> <3,0,-4> }
        triangle { <-3,2,0> <-3,1,0> <1,1,5> }
        texture { MyTexture }
}
```



Anwendung von Meshes

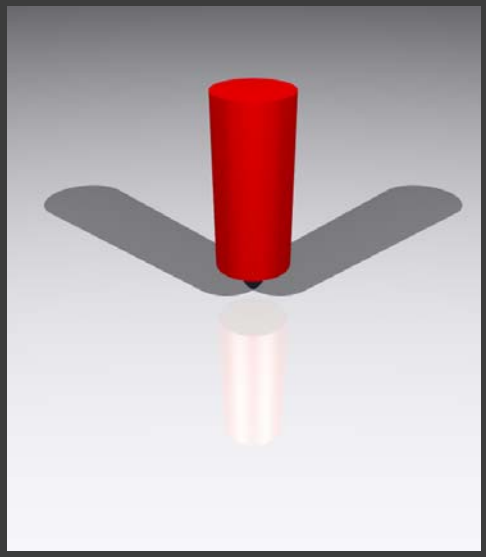
- 3D-Echtzeitgraphik basiert aktuell auf Dreiecken
  - Beschleunigerkarten unterstützen gezielt Dreiecke
  - Siehe DirectX und OpenGL (nächste Woche)
- Identische Arbeitsschritte
  - Übertragung einer Szene an Graphikkarte
  - Generierung einer Povray-Beschreibung aus Dreiecken
- Erweitere Definition: mesh2
  - Anlegen von Knotenvektoren
  - Definition der Dreiecke über Indizes
  - Einsparungen bei "zusammenhängenden" Meshes
    - Knoten tauchen immer in mehreren Dreiecken auf

CSG: Beispiel 1

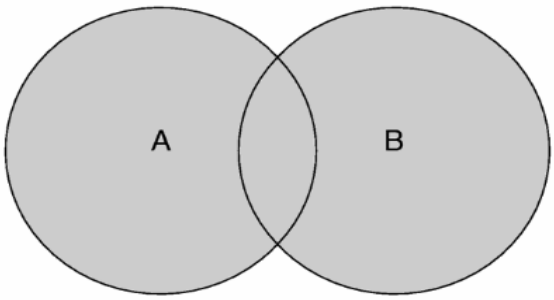
- Beschreibung:

```
cylinder {
        <0,1,0> <0,6,0> 1
        texture { MyTexture }
}
```

- Weitere Bestandteile
  - Zwei Lichtquellen
  - Reflektiver Untergrund
  - Kamera



Union



The union of two objects

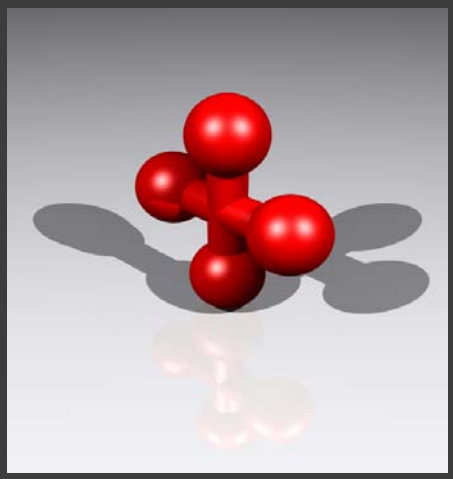
The simplest kind of CSG is the `union`. The syntax is:

```
UNION:
    union
    {
        OBJECTS...
        [OBJECT_MODIFIERS...]
    }
```

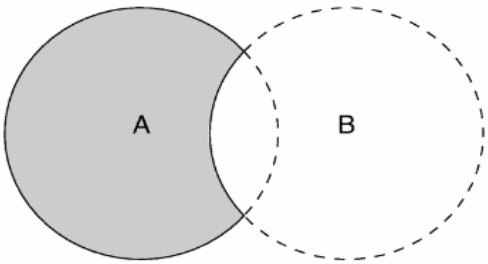
CSG: Beispiel 2

- Kugeln an Zylinderenden hinzufügen
- Beschreibung:

```
#declare Sol = union {
        cylinder { <0,1,0> <0,5,0> 0.5 }
        sphere { <0, 1, 0> 1 }
        sphere { <0, 5, 0> 1 }
        texture { MyTexture }
}

object { Sol }
object { Sol
        translate <0,-3,0>
        rotate <90,0,0>
        translate <0,3,0>
}
```



Difference

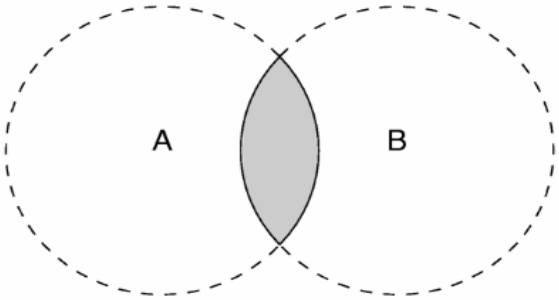


The difference between two objects

The syntax is:

```
DIFFERENCE:
    difference
    {
        SOLID_OBJECTS...
        [OBJECT_MODIFIERS...]
    }
```

Intersection



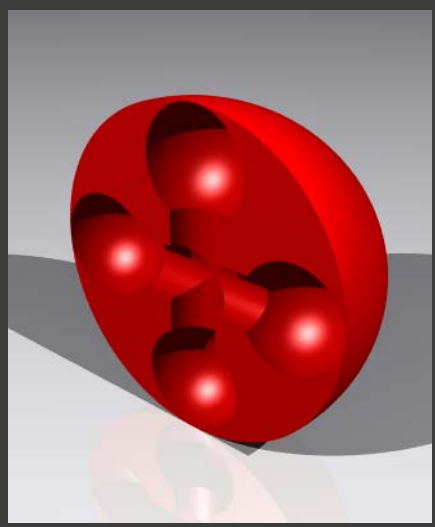
The intersection of two objects

The syntax is:

```
INTERSECTION:
    intersection
    {
        SOLID_OBJECTS...
        [OBJECT_MODIFIERS...]
    }
```

CSG: Beispiel 3

```
#declare Sol = union {
        cylinder { <0,1,0> <0,5,0> 0.5 }
        sphere { <0, 1, 0> 1 }
        sphere { <0, 5, 0> 1 }
        texture { MyTexture }
}

#declare TwoSol = union {
        object { Sol }
        object { Sol
                translate <0,-3,0>
                rotate <90,0,0>
                translate <0,3,0>
        }
}

#declare HalfSphere = intersection {
        sphere { <0,3,0> 3 }
        box { <0,0,-3> <-3,6,3> }
}

difference {
        object { HalfSphere }
        object { TwoSol }
        texture { MyTexture }
}
```



Object Modifiers

- Material-, Reflektions, Refraktionseigenschaften etc.

```
OBJECT_MODIFIER:
    clipped_by { UNTEXTURED_SOLID_OBJECT... } |
    clipped_by { bounded_by }                 |
    bounded_by { UNTEXTURED_SOLID_OBJECT... } |
    bounded_by { clipped_by }                 |
    no_shadow                                 |
    no_image [ Bool ]                         |
    no_reflection [ Bool ]                    |
    inverse                                   |
    sturm [ Bool ]                            |
    hierarchy [ Bool ]                        |
    double_illuminate [ Bool ]                |
    hollow  [ Bool ]                          |
    interior { INTERIOR_ITEMS... }                                  |
    material { [MATERIAL_IDENTIFIER][MATERIAL_ITEMS...] } |
    texture { TEXTURE_BODY }                  |
    interior_texture { TEXTURE_BODY } |
    pigment { PIGMENT_BODY }                  |
    normal { NORMAL_BODY }                    |
    finish { FINISH_ITEMS... } |
    photons { PHOTON_ITEMS...}
    TRANSFORMATION
```

Pigment

- Farben und Farbmuster

```
PIGMENT:
    pigment {
        [PIGMENT_IDENTIFIER]
        [PIGMENT_TYPE]
        [PIGMENT_MODIFIER...]
    }
PIGMENT_TYPE:
    PATTERN_TYPE | COLOR |
    image_map {
        BITMAP_TYPE "bitmap.ext" [IMAGE_MAP_MODS...]
    }
PIGMENT_MODIFIER:
    PATTERN_MODIFIER | COLOR_LIST | PIGMENT_LIST |
    color_map { COLOR_MAP_BODY } | colour_map { COLOR_MAP_BODY } |
    pigment_map { PIGMENT_MAP_BODY } | quick_color COLOR |
    quick_colour COLOR
```

- Einfache Farbe
  - texture { pigment { color rgb<1,0,0> } }

Image Map

```
IMAGE_MAP:
    pigment
    {
        image_map
        {
            [BITMAP_TYPE] "bitmap[.ext]"
            [IMAGE_MAP_MODS...]
        }
        [PIGMENT_MODFIERS...]
    }
BITMAP_TYPE:
    gif | tga | iff | ppm | pgm | png | jpeg | tiff | sys
IMAGE_MAP_MOD:
    map_type Type | once | interpolate Type |
    filter Palette, Amount | filter all Amount |
    transmit Palette, Amount | transmit all Amount
```

After the optional *BITMAP_TYPE* keyword is a string expression containing the name of a bitmapped image file of the specified type. If the *BITMAP_TYPE* is not given, the same type is expected as the type set for output.
Example:

```
plane {
  -z,0
  pigment {
    image_map {png "Eggs.png"}
  }
}

plane {
  -z,0
  pigment {
    image_map {"Eggs"}
  }
}
```

The second method will look for, and use "Eggs.png" if the output file type is set to be png (Output_File_Type=N in INI-file or +FN on command line). It is particulary usefull when the image used in the image_map is also rendered with POV-Ray.

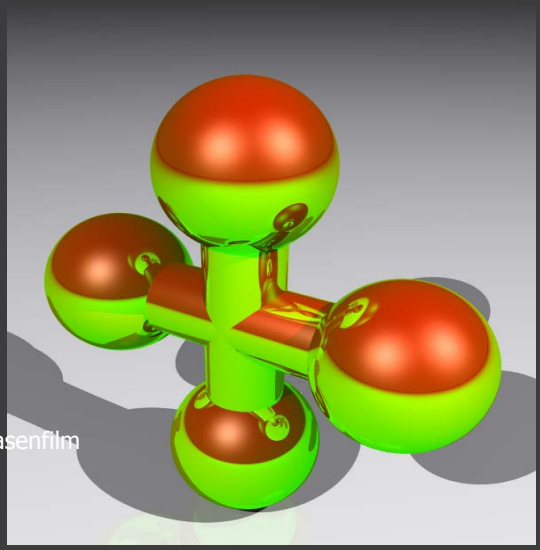
Finish

- "Polieren" der Objektoberfläche
  - Eigenleuchten (Ambient)
  - Reflektionseigenschaften
    - Lichtstreuung
    - Brillianz
    - Widerschein von Lichtquellen (Phong, Specular, Diffuse)
  - Eingeschränkt Oberflächenstrukturen (Roughness)

```
FINISH:
    finish { [FINISH_IDENTIFIER] [FINISH_ITEMS...] }
FINISH_ITEMS:
    ambient COLOR | diffuse Amount | brilliance Amount |
    phong Amount | phong_size Amount | specular Amount |
    roughness Amount | metallic [Amount] | reflection COLOR |
    crand Amount | conserve_energy BOOL_ON_OF |
    reflection { Color_Reflectin_Min [REFLECTION_ITEMS...] }|
    irid { Irid_Amount [IRID_ITEMS...] }
REFLECTION_ITEMS:
    COLOR_REFLECTION_MAX | fresnel BOOL_ON_OFF |
    falloff FLOAT_FALLOFF | exponent FLOAT_EXPONENT |
    metallic FLOAT_METALLIC
IRID_ITEMS:
    thickness Amount | turbulence Amount
```

Finish: Beispiele

```
#declare MyTexture = texture {
        pigment {
                color Red
        }
        finish {
                ambient 0.2
                phong 0.5 phong_size 10
                reflection { 0.01 }
                diffuse 0.4
//                specular 0.5
//                roughness 0.5
//                metallic 0.5
                reflection Green
//                irid { 0.2 }   // Seifenblasenfilm
        }
}
```



Brechung

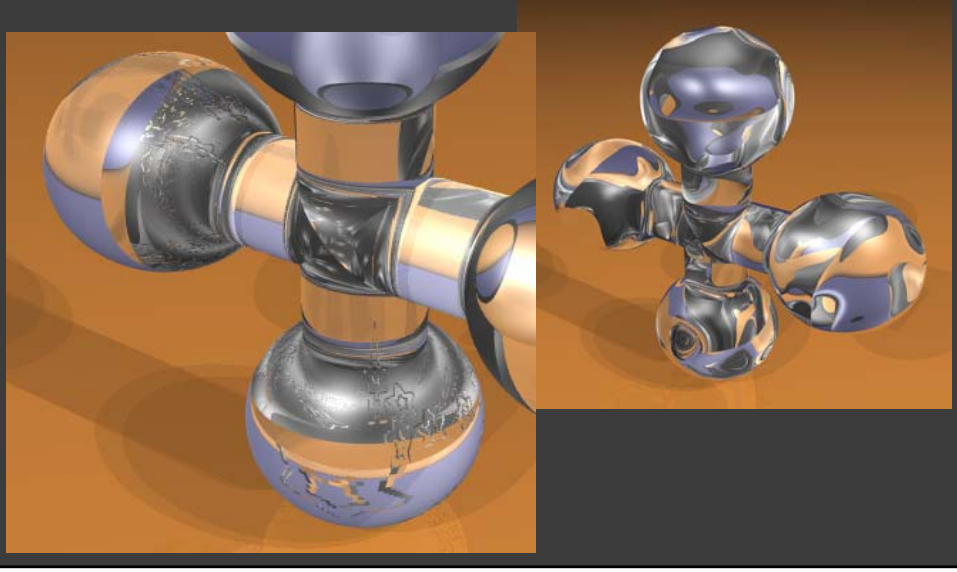
- Interior-Eigenschaften bei Objekten
- ior = Brechungsindex
  - 1.0 Luft
  - 1.33 Wasser
  - 1.5 Glas
  - 2.4 Diamand
- Zusätzliche Farbeigenschaft
  - Filter
  - Trasmit

Weitere Möglichkeiten bzgl. Oberfläche

- Trick zur Veränderung der Oberflächenbeschaffenheit
- Störungen bzgl. Der Oberflächennormale
  - Parameter
  - Bump Map
- Objektform ändert sich dadurch nicht
  - Diskrepanz zwischen Silhouette und scheinbar rauher Oberflächenstruktur

```
NORMAL:
  normal { [NORMAL_IDENTIFIER] [NORMAL_TYPE] [NORMAL_MODIFIER...] }
NORMAL_TYPE:
  PATTERN_TYPE Amount |
  bump_map { BITMAP_TYPE "bitmap.ext" [BUMP_MAP_MODS...]}
NORMAL_MODIFIER:
  PATTERN_MODIFIER | NORMAL_LIST | normal_map { NORMAL_MAP_BODY } |
  slope_map{ SLOPE_MAP_BODY } | bump_size Amount |
  no_bump_scale Bool | accuracy Float
```

Bumps: Beispiel



Lichtquellen

```
LIGHT_SOURCE:
    light_source
    {
        <Location>, COLOR
        [LIGHT_MODIFIERS...]
    }
LIGHT_MODIFIER:
    LIGHT_TYPE | SPOTLIGHT_ITEM | AREA_LIGHT_ITEMS |
    GENERAL_LIGHT_MODIFIERS
LIGHT_TYPE:
    spotlight | shadowless | cylinder | parallel
SPOTLIGHT_ITEM:
    radius Radius | falloff Falloff | tightness Tightness |
    point_at <Spot>
PARALLEL_ITEM:
    point_at <Spot>
AREA_LIGHT_ITEM:
    area_light <Axis_1>, <Axis_2>, Size_1, Size_2 |
    adaptive Adaptive | jitter Jitter | circular | orient
GENERAL_LIGHT_MODIFIERS:
    looks_like { OBJECT } |
    TRANSFORMATION fade_distance Fade_Distance |
    fade_power Fade_Power | media_attenuation [Bool] |
    media_interaction [Bool] | projected_through
```

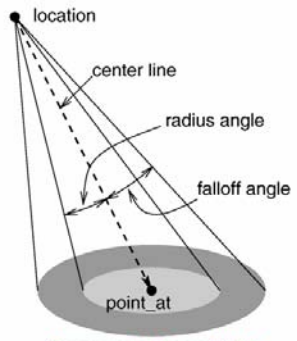
Spot Light

```
SPOTLIGHT_SOURCE:
    light_source
    {
        <Location>, COLOR spotlight
        [LIGHT_MODIFIERS...]
    }
LIGHT_MODIFIER:
    SPOTLIGHT_ITEM | AREA_LIGHT_ITEMS | GENERAL_LIGHT_MODIFIERS
SPOTLIGHT_ITEM:
    radius Radius | falloff Falloff | tightness Tightness |
    point_at <Spot>
```

Default values:

```
radius:    30 degrees
falloff:   45 degrees
tightness:  0
```

The `point_at` keyword tells the spotlight to point at a particular 3D coordinate. A line from the location of the spotlight to the `point_at` coordinate forms the center line of the cone of light. The following illustration will be helpful in understanding how these values relate to each other.



The geometry of a spotlight

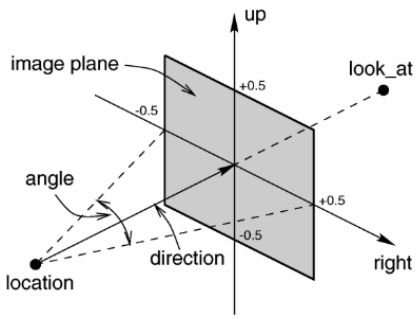
The `falloff`, `radius`, and `tightness` keywords control the way that light tapers off at the edges of the cone. These four keywords apply only when the `spotlight` or `cylinder` keywords are used.

Camera

```
CAMERA:
    camera{ [CAMERA_ITEMS...] }
CAMERA_ITEM:
    CAMERA_TYPE | CAMERA_VECTOR | CAMERA_MODIFIER |
    CAMERA_IDENTIFIER
CAMERA_TYPE:
    perspective | orthographic | fisheye | ultra_wide_angle |
    omnimax | panoramic | cylinder CylinderType | spherical
CAMERA_VECTOR:
    location <Location> | right <Right> | up <Up> |
    direction <Direction> | sky <Sky>
CAMERA_MODIFIER:
    angle HORIZONTAL [VERTICAL] | look_at <Look_At> |
    blur_samples Num_of_Samples | aperture Size |
    focal_point <Point> | confidence Blur_Confidence |
    variance Blur_Variance | NORMAL | TRANSFORMATION
```

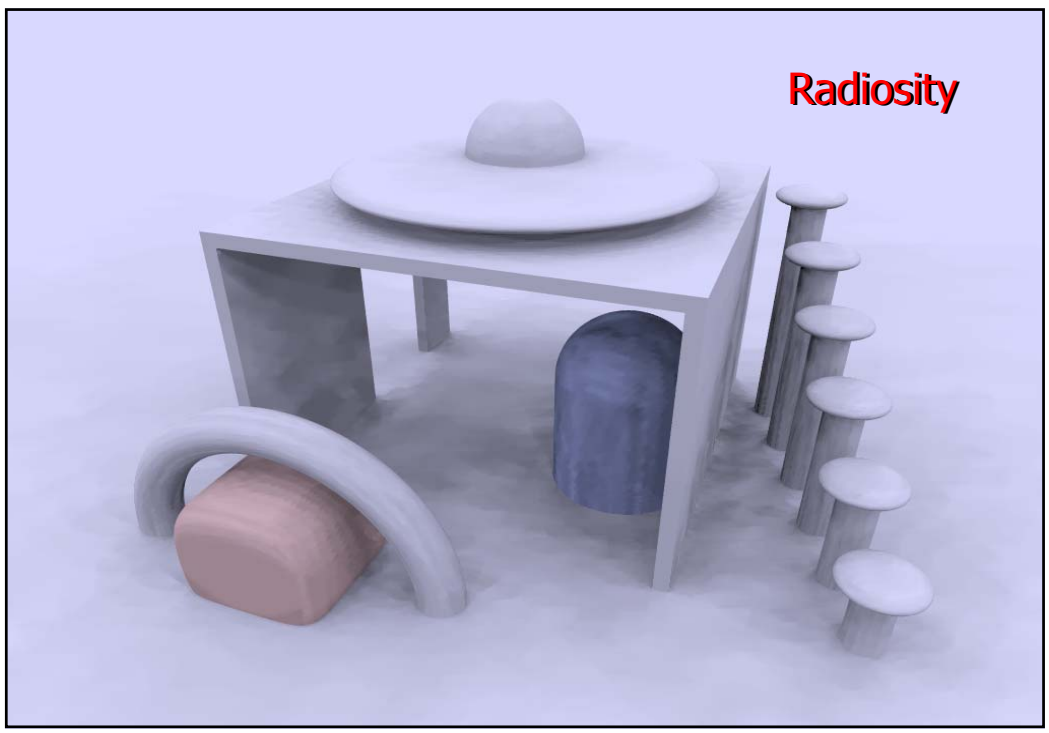
Camera default values:

```
DEFAULT CAMERA:
  camera {
    perspective
    location <0,0,0>
    direction <0,0,1>
    right 1.33*x
    up y
    sky <0,1,0>
  }

CAMERA TYPE: perspective
angle      : ~67.380 ( direction_length=0.5*
                       right_length/tan(angle/2) )

confidence : 0.9 (90%)
direction  : <0,0,1>
focal_point: <0,0,0>
location   : <0,0,0>
look_at    : z
right      : 1.33*x
sky        : <0,1,0>
up         : y
variance   : 1/128
```



The perspective camera

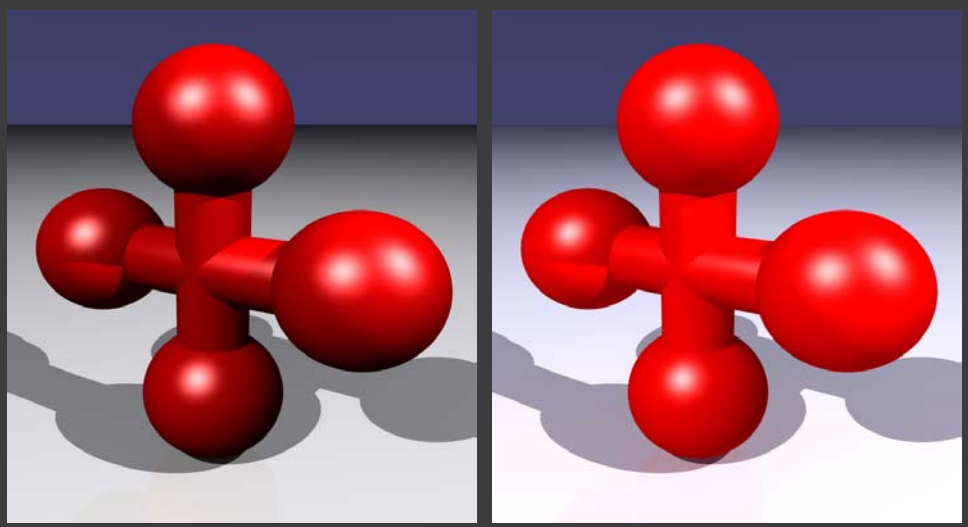
Radiosity



Radiosity

- Traditionelle Raytracer bestimmen die Helligkeit an jedem Punkt über
  - Diffuse (Lichtzugewandte Seite wird heller)
  - Specular (Widerschein der Lichtquelle)
  - Reflektion
  - Ambient (Grundhelligkeit)
- Radiosity
  - Ambient wird genauer berechnet
  - In welchem Maß und und mit welchen Farben "strahlen" benachbarte Objekte

Radiosity: Beispiel



Normal: 5 Sekunden

Radiosity: 35 Sekunden