VISUAL COMPUTING

Kapitel 1 – Transformationen

	2D	3D	Inverse 3D
Translation	$T_{trans} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$	$T_{trans} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$T_{trans}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Skalierung	$T_{sc} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$T_{sc} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$T_{sc}^{-1} = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Spiegelung	$T_{sp} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$T_{sp} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$T_{sp}^{-1} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Rotation	$T_{rot} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$T_{x-rot} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $T_{y-rot} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $T_{z-rot} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\alpha = -\alpha$ Retourdrehen

Berechnung / Reihenfolge

Matrix/Matrizen anwenden	$u' = T * u$	$v' = T_2\,T_1 * v$	$w' = T_3\,T_2\,T_1 * w$
Matrix/ Matrizen ausrechnen (Reihenfolge beachten) T_1 vor T_2 vor T_3	$\begin{array}{c\|c} & u \\ \hline T & u' \end{array}$	$\begin{array}{c\|cc} & T_1 & v \\ \hline T_2 & T_{21} & v' \end{array}$	$\begin{array}{c\|ccc} & T_2 & T_1 & w \\ \hline T_3 & T_{32} & T_{321} & w' \end{array}$
Transformation und Rücktransformation $P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$	$P' = T_{x-rot} * T_{trans} * P$ Erst Translation, dann Rotation	$P = T_{x-rot}^{-1} * T_{trans}^{-1} * P'$ Erst Gegentranslation, dann Gegenrotation	
Homogene Transformation	Objekt behält Seitenverhältnisse etc.		
Nicht-Homogene Transformation	Z.B. einseitige Skalierung		

Vektor zu einem Richtungsvektor erweitern	Vektor zu einem Raumpunkt erweitern	3x3 Matrix zu 4x4 Matrix erweitern
$\begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} \rightarrow \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ 0 \end{pmatrix}$	$\begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} \rightarrow \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ 1 \end{pmatrix}$	$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \rightarrow \begin{bmatrix} a & b & c & dx \\ d & e & f & dy \\ g & h & i & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Kapitel 2 – Rasterung von Strecken - Bresenham-Rasterisierungsalgorithmus

Prinzip	Erzeugung der Rasterpunkte durch Rundung		
Algorithmus	$E_1 = \frac{\Delta Y}{\Delta X} - \frac{1}{2}$	falls $E_1 > 0$, nächst höherer Punkt (aufrunden)	$E = E + \frac{\Delta Y}{\Delta X} - 1$
		falls $E_1 \leq 0$, nächst tieferer Punkt (abrunden)	$E = E + \frac{\Delta Y}{\Delta X}$
Verbesserung	Da nur die Vorzeichen von E interessieren -> mit Mulitplikation von $2 * \Delta X$ die Quotienten entfernen. $E_1 = 2 * \Delta Y - \Delta X$		
Resultat	Algorithmus, der nur mit ganzen Zahlen, Addition, Subtraktion und Shift (Shift um 1 = Division mit 2) arbeitet.		

Variationen

Variante 1 - mit Steigung s

```
dx=x2-x1; dy=y2-y1;
x=x1; y=y1;
s=dy/dx;
while(x <= x2){
   plot(x,y);
   y+=s;
   x++;
}
```

x,y,s als double

Variante 2 - mit Entscheidung E

```
dx=x2-x1; dy=y2-y1;
x=x1; y=y1;
s=dy/dx;
while(x <= x2){
   plot(x,y);
   x++;
   d+=s;
   if(d>0.5){
      y++;
      d--;
   }
}
```

x, y als int
s,d als float

Variante 3 - mit *dx, *2

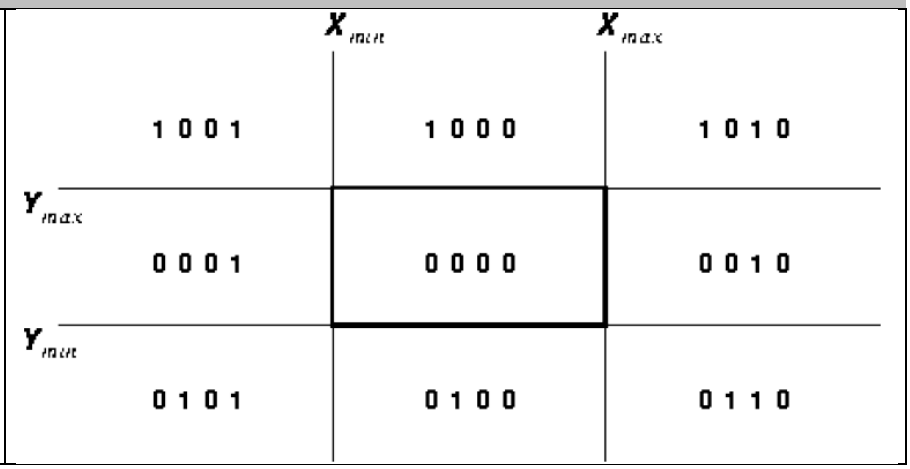
```
dx=x2-x1; dy=y2-y1;
x=x1; y=y1;
s=2*dy;
while(x <= x2){
   plot(x,y);
   x++;
   d+=s;
   if(d>dx){
      y++;
      d-=dx*2;
   }
}
```

x,y,s,d als int

Kapitel 3 – Clipping (Abschneidung)

Cohen Sutherland Algorithmus

Es soll eine Linie gezeichnet werden, es ist jedoch nicht bekannt ob die Linie auf dem Bildschirm überhaupt sichtbar ist. Es soll nur der Teil gezeichnet werden, welcher auch auf dem Schirm sichtbar ist.



- Der Sektor 0000 entspricht dem sichtbaren Bildschirm
- Alle anderen Sektoren befinden sich ausserhalb des sichtbaren Bereichs

4bit Kodierung (Outcode)

falls $y > y_{max}$ dann Bit 4=1	falls $y < y_{min}$ dann Bit 3=1	falls $x > x_{max}$ dann Bit 2=1	falls $x < x_{min}$ dann Bit 1=1

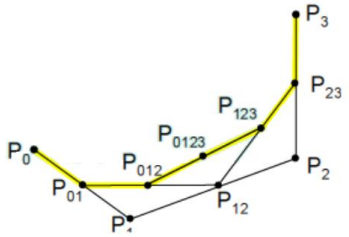
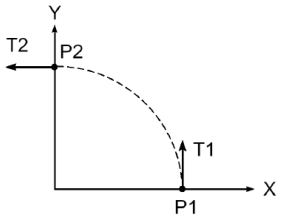
Vorgehen

1. Für den Anfangs- und den Endpunkt soll die Kodierung generiert werden
2. Nun gilt es herauszufinden wie die Linie auf dem Bildschirm steht:
   - Linie ist nicht auf dem Bildschirm: (code1 & code2) > 0
   - Linie ist komplett auf dem Bildschirm: (code1 | code2) == 0
   - Linie ist teilweise auf dem Bildschirm:
     Bei ungleichem Wert einer Bitstelle wird mit der entsprechenden Fenstergrenze geschnitten.
     Vektor aufteilen (so wird ein Teil eliminiert), evt. 2ter Durchlauf

Wraparound = Überlauf der Koordinatenadressierung

Kapitel 4 – Kurven

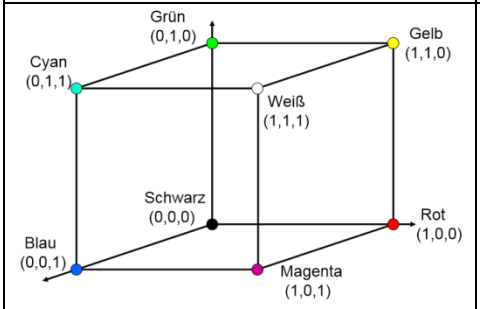
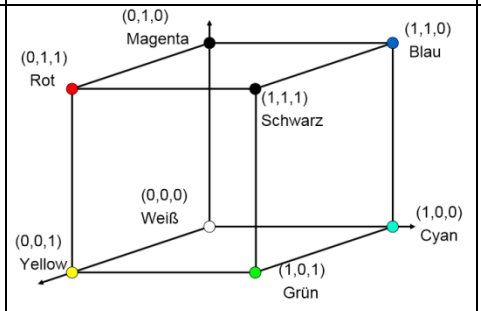
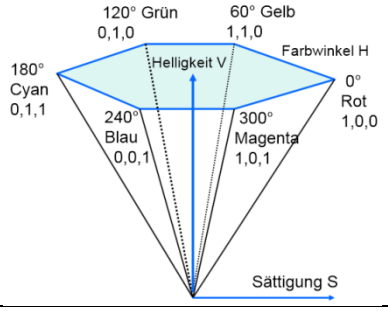
Formen	**Explizite Form**	$y = ax + b, \quad Gerade$ $y = ax^2 + bx + c, \quad Parabel$
	Implizite Form	$(x - x_1)^2 + (y - y_1)^2 = R^2, \quad Kreis$ $Ax + By + Cz = 0, \quad Ebene$
	Parameter-Form	$x = a + R\cos t$ $y = b + R\sin t$

Bézier	Wie Hermite zu lösen, jedoch mit 4 Punkten: $\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} \vec{P_1} \\ \vec{P_2} \\ \vec{P_3} \\ \vec{P_4} \end{bmatrix} = \begin{bmatrix} \vec{a} \\ \vec{b} \\ \vec{c} \\ \vec{d} \end{bmatrix}$ $\vec{F}(t) = \vec{a}t^3 + \vec{b}t^2 + \vec{c}t + \vec{d}$	Mit Casteljau Algorithmus gelöst (Anhang) 
Splines > Hermite	Zwei Punkte, zwei definierte Steigungen  $\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} \vec{P_1} \\ \vec{P_2} \\ \vec{T_1} \\ \vec{T_2} \end{bmatrix} = \begin{bmatrix} \vec{a} \\ \vec{b} \\ \vec{c} \\ \vec{d} \end{bmatrix}$	Vier Bedingungen -> Polynom 3ten Grades $\vec{F}(t) = \vec{a}t^3 + \vec{b}t^2 + \vec{c}t + \vec{d}$ 1. $\vec{F}(0) = P_1 = d$ 2. $\vec{F}(1) = P_2 = \vec{a} + \vec{b} + \vec{c} + \vec{d}$ 3. $\vec{F}'(0) = T_1 = \vec{c}$ 4. $\vec{F}'(1) = T_2 = 3\vec{a} + 2\vec{b} + \vec{c}$
Splines > Basic Splines (B-Splines)	**Koeffizientenarray** $T = \left[\underbrace{0, 0, \ldots,}_{k\ mal} \underbrace{1, 2, \ldots, n-1}_{inkrementieren}, \underbrace{n, n, \ldots}_{k-mal}\right]$ • n = Anzahl Stützpunkte -1 • k = Grad des Polynoms - 1 **Punkte (t = 0 => Startpunkt, t=1 => Endpunkt)** $r(t) = \sum_{i=0\ldots n} \left(N_{i,k}(t) * r_i\right)$ **Beispiel:** $r(0.5) = N_{0,3}(0.5) * r_0 + N_{1,3}(0.5) * r_1 + N_{2,3}(0.5) * r_2$ **Koeffizientenberechnung:** $N_{i,k}(t) = \frac{t - T[i]}{T[i+k-1] - T[i]} * N_{i,k-1}(t) * \frac{T[i+k] - t}{T[i+k] - T[i]} * N_{i+1,k-1}(t)$ $N_{i,1}(t) = \begin{cases} 1\ wenn\ T[i] \le t < T[i+1] \\ 0\ sonst \end{cases}$	

kapitel 5 – Quaternionen

Komplexe Zahlen	$z_1 = r_1 * e^{i\varphi_1} = r_1(\cos\varphi_1 + i * \sin\varphi_1)$ $z_2 = r_2 * e^{i\varphi_2} = r_2(\cos\varphi_2 + i * \sin\varphi_2)$ $z_1 * z_2 = r_1 * r_2 * e^{i(\varphi_1+\varphi_2)}$
Idee der Quaternionen	$r_2 = 0 \rightarrow z_1 * z_2 = r_1 * e^{i(\varphi_1+\varphi_2)}$ Mit Komplexen Zahlen kann ich sehr schnell Rotationen machen -> Erweiterung der Komplexen Zahlen auf 3D.
Definition	$q = r + xi + yj + zk$ $q = [s, \vec{v}]$ s = Skalar, v = Vektor, Es gilt, s = r und v = $\begin{matrix} x \\ y \\ z \end{matrix}$
Grundoperationen	Addition: $q_1 + q_2 = [s_1 + s_2, \overrightarrow{v_1} + \overrightarrow{v_2}]$ Multiplikation mit Skalar: $cq = [cs, cv]$ Multiplikation: $q_1 * q_2 = [s_1, s_2 - v_1v_2, s_1v_2 + s_2v_1 + s_2v_1 + v_1 \times v_2]$ Drehung mit normiertem Quaternion um Winkel γ: Definition Achse v: $q = [\cos(\gamma/2), \sin(\gamma/2) * v]$ mit $\lvert v\rvert = 1$ Rotation von p um v mit Winkel γ: $p' = q * p * q'$

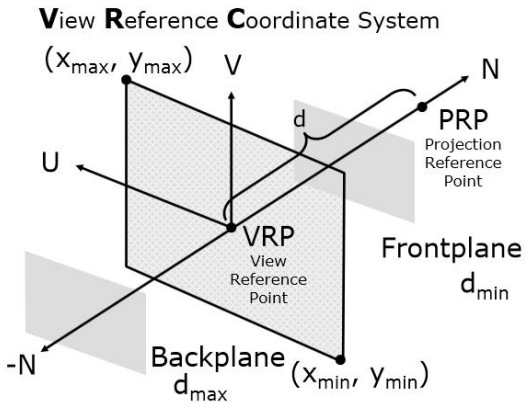
Kapitel 6 – Farben

RGB	CMY	HSV
		

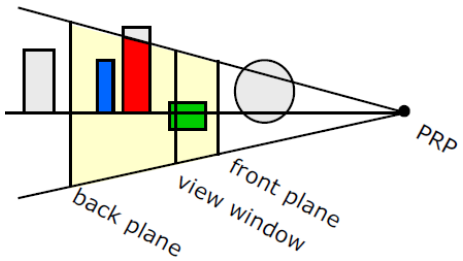
$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Kapitel 7 – Viewing Pipeline

Synthetische Kamera



- Sicht von rechts oben
- Beschreibung durch:
  $d, x_{max}. y_{max}, x_{min}, y_{min}, d_{min}, d_{max}$
- Nur Gegenstände zwischen diesen beiden Schichten werden dargestellt.



Ablauf

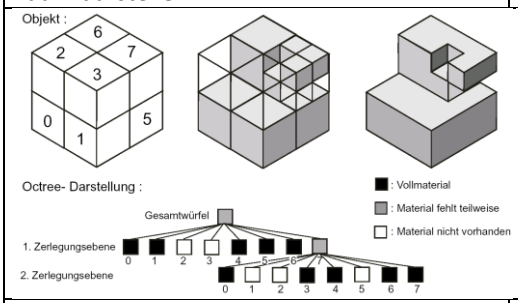
1. Modelling:
   $$MC\ (Modellkoordinate) \rightarrow WC(Weltkoordinate)$$
2. View Orientation
   $$WC \rightarrow VRC\ (Kamerakoordinaten)$$
3. View Mapping
   $$VRC \rightarrow NPC\ (in\ Einheitswürfel\ überführen)$$
4. Device Mapping
   $$NPC \rightarrow DC\ (auf\ Bildschirm\ projizieren)$$

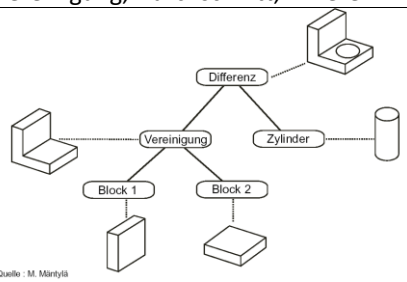
$$Bildschirmkoordinate = T_{device} * T_{viewmap} * T_{viewref} * T_{model} * Punkt$$

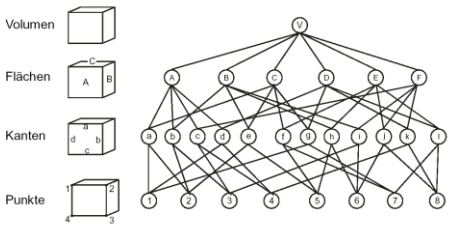
Kapitel 8 – Geometriemodell & Sichtbarkeit

back-face culling	Rückseiten müssen nicht gerendert werden Erkennung: Normalenvektor in Bildschirmkorrdinaten hat eine positive z-Komponente
Painters	Polygon mit grösstem z-Wert zuerst malen 1. Sortiere Polygone nach z-Wert 2. Bei Überlappungen Schnittpolygone berechnen 3. Beginne mit dem Zeichnen
Punktorientierte Verfahren	**z-Buffer** Jedes Pixel hat einen z-Wert, falls dieser kleiner ist als der aktuelle z-Wert, Bildspeicher überschreiben keine Transparenz, Abtastfehler wenn höhere Auflösung, beschränkte Genauigkeit

Octree	**CSG (Constructive Solid Geometry)**	**BREP**
Würfel in 8 Teile aufteilen und in einem Baum darstellen	Volumenprimitiva und Operationen: Vereinigung, Durchschnitt, Differenz	Beschreibung des Körpers durch Randflächen: Ecke-Kanten-Flächen-Körper







Euler'sche Gleichung

$$E - K + S - I = 2(Z - G)$$

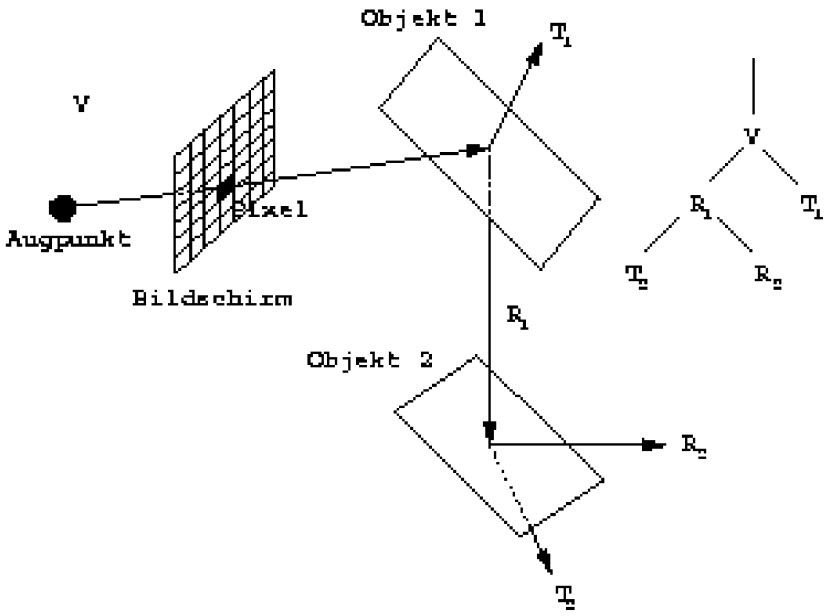
E	Ecken
K	Kanten
S	Seiten
I	innere Zyklen
Z	Komponenten
G	Löcher

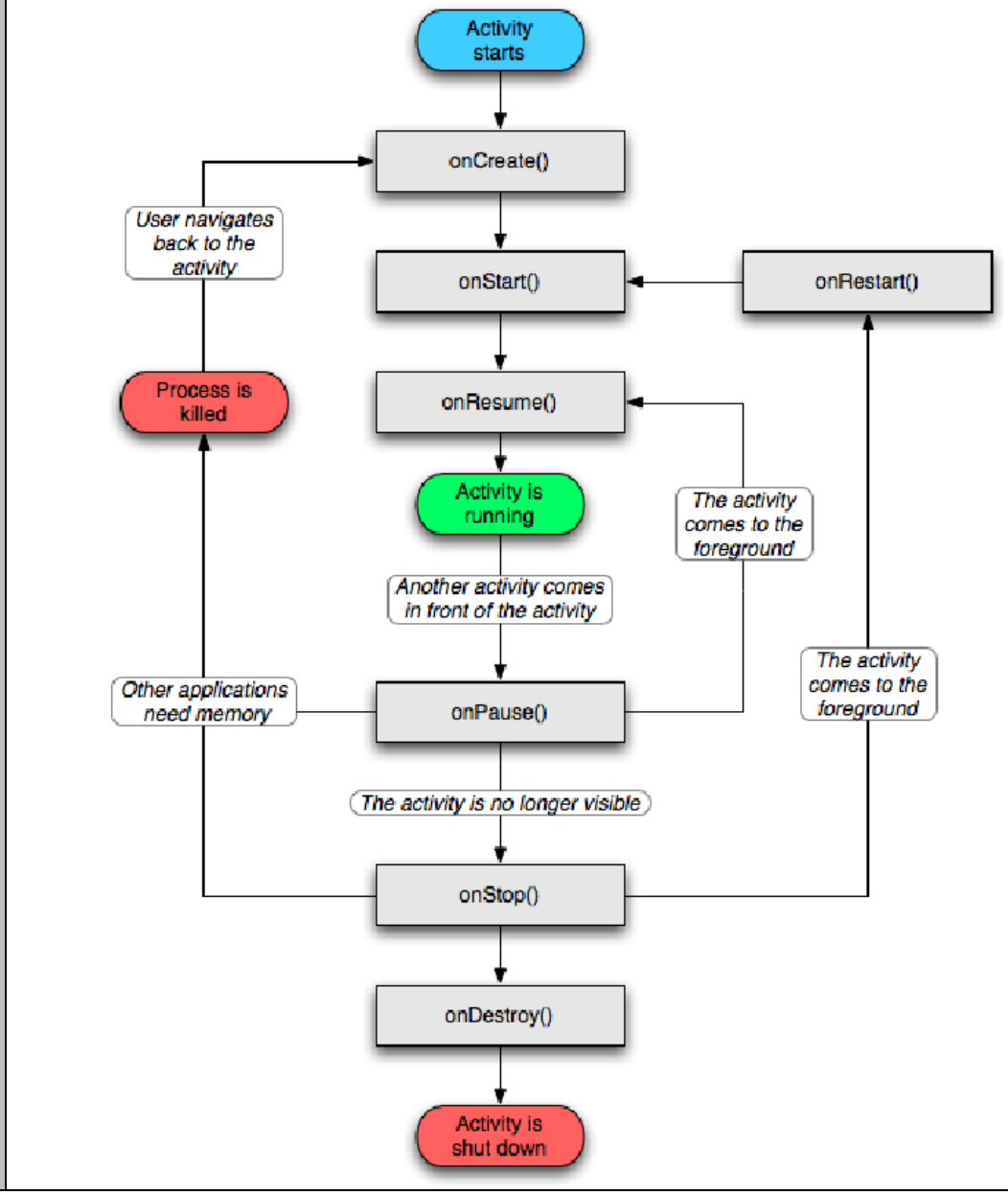
Kapitel 9 – JOGL / OpenGL

JOGL	Java OpenGL	
OpenGL	Konkurrent zu DirectX (Windows) plattformunabhängig, funktionsorientiert grundlegende geometrische Objekte, Transformationen, Bilddarstellung (Rendering), ...	
Code	Linie(n)	
	Farben	`void GL.glColor3f(float red, green, blue);`
Anbindungen an JAVA	in AWT	`GLCanvas`
	in Swing	`GLJPanel`
Methoden	`init();`	Zur Initialisierung von Hilfsklassen, wird am Anfang aufgerufen
	`dispose();`	Um Resourcen freizugeben, wird am Ende aufgerufen
	`display();`	Zum zeichnen der Objekte (Dreiecke, Linien)
	`reshape();`	bei einem repaint, für Ansichtsbereich, Projektionsmodus und das Anzeigevolumen
WebGL	OpenGL für die Browser (z.B. Chrome, Firefox) ▪ keine Installation ▪ kein .NET Framework ▪ verzichten auf Sicherheitsaspekte ▪ Javascript Was WebGL von herkömmlichen Javascript (2D) Grafikfunktionen unterscheidet ist, dass ein Großteil der erforderlichen Berechnungen auf der Grafikarte statt im Hauptprozessor ausgeführt wird (GPU statt CPU). Die GPU (Graphics Processing Unit) ist für grafische Berechnungen ausgelegt und hat v.A. gegenüber der CPU den Vorteil dass sie viele Rechenoperationen GLEICHZEITIG berechnen kann.	
Silverlight	in C#	

Kapitel 10 – Shading + Raytracing

Definition	**Shading** oder **Schattierung** ist ein Begriff aus der 3D-Computergrafik, der im weiten Sinn die Simulation der Oberflächeneigenschaften von Objekten bezeichnet. Im Speziellen bezeichnet Shading die Anwendung eines Interpolationsverfahrens, mit dem der Normalenvektor auf beliebigen Punkten eines Polygonnetzes berechnet wird. Interpolative Shading-Techniken können dazu verwendet werden, um Oberflächen „glatter“ aussehen zu lassen. (Wikipedia)
Flat Shading	Flat Shading oder Constant Shading ist ein sehr einfaches Schattierungsverfahren. • Jedes Pixel eines Polygons anhand der Flächennormale die gleiche Farbe bzw. den gleichen Lichtwert. • Flat Shading findet anwendung bei Objekten mit Flächen (Quader, Würfel etc.)
Gouraud Shading	schnell, bei grossen Polygonen Ungenauigkeiten, diffuse Reflexion **Farbberechnung** 1. Zunächst werden für das Polygon die darzustellenden Farben an den **Eckpunkten (Vertices)** berechnet. (Details unter Beleuchtung) 2. Nun wird das Polygon auf die Bildebene Projiziert. 3. Die entstandene zweidimensionale Fläche wird nun Zeilenweise abgearbeitet. a. Die Kanten des Polygons werden aus den Eckpunkten interpoliert. b. Die inneren Punkte werden aus den Kanten interpoliert. Zur Berechnung des Farbwertes eines **Vertex** werden die **Flächennormalen** der anliegenden Flächen **gemittelt**. Die von der Fläche wiedergegebene Farbe wird aus dem diffusen Reflexionskoeffizienten, aus der Lichtquelle sowie aus dem Winkel zwischen Flächennormale und Lichtstrahl zur Lichtquelle gebildet.
Phong Shading	Bei diesem Verfahren werden ebenfalls die Normalen für die Eckpunkte interpoliert. Jedoch werden für die **Kantenpixel** nicht die Farbwerte interpoliert, sondern ebenfalls die **Normalenvektoren**. Dies bedeutet einen erheblich grösseren Rechenaufwand. Die Qualität ist entsprechend besser. Es gibt weiche Übergänge zwischen den Polygonen.
Raytracing	Es wird dem Lichtstrahl gefolgt, ausgehend vom Betrachter, und bei jedem Schnitt mit einem Objekt der reflektierte und ideal gebrochene Strahl weiterverfolgt. Vorgehen für jeden Pixel des Bildschirms: 1. Bestimme nächstliegenden Schnittpunkt des Sehstrahls mit einem Objekt der Szene. 2. Berechne ideal reflektierten Lichtstrahl. 3. Berechne die Leuchtdichte aus dieser Richtung. 4. Berechne ideal gebrochenen Lichtstrahl. 5. Berechne die Leuchtdichte aus dieser Richtung. 6. Werte das Phongbeleuchtungsmodell im Schnittpunkt aus und addiere die gewichtete Leuchtdichte der sekundärstrahlen.



Kapitel 11 – Augmented Reality (Erweitere Realität)	
Definition	Erweiterung der Realitätswahrnehmung mit Bildern oder Videos mit computergenerierten Zusatzinformationen
Android	

Kapitel 12 – Physikalische Modellierung	
Starre Körper	hat Masse, Trägheitsmoment, Bewegungszustand, Kollisionsgeometrie
Constraints (Joints)	damit können Körper verbunden werden (Scharniere, Kugellager)