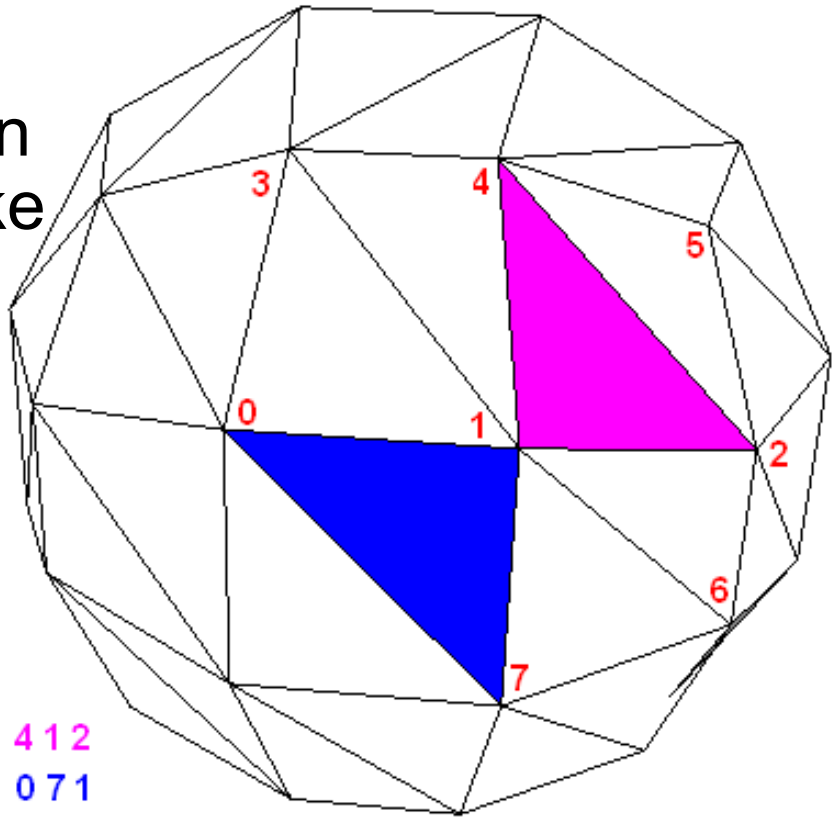
Übersicht

- Mesh
- Beleuchtung
- Texturen
- Transparenz
- Mipmaps
- LOD
- Nachtdarstellung
- Animationen
- 3D-Editor

Mesh-Aufbau

- Komplexer Aufbau, kann mehrere 100 Vertices enthalten
- Vertices werden durchnumeriert und über Indexgruppen werden die Dreiecke definiert
- Ein Meshsubset muß keine zusammenhängende Fläche bilden
- Ein Meshubset hat für alle Dreiecke dieselben Einstellungen (Textur, Farbe, Filter usw.)
- Meshsubset ist in sich starr
- Meshsubset wird immer ganz oder gar nicht gezeichnet



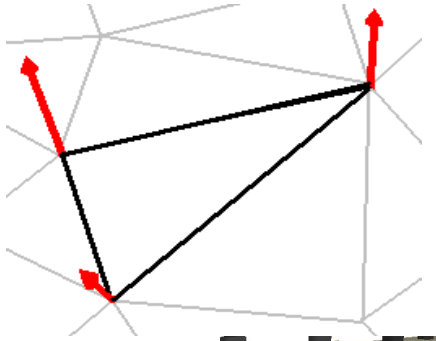
- Meshsubset wird in einem Durchgang gerendert -> sehr schnell (Optimale Vertices/sek. bei ca. 4000 Vertices/Mesh)
- Ein großes Meshsubset ist effizienter als viele kleine

Rendervorgang

- 1. Schritt: Transformation der Vertices vom 3D auf die Bildschirmfläche. Damit ist Lage und Beleuchtungseinfluß für jeden Vertex ermittelt

- 2. Schritt: Ausfüllen der Dreiecksflächen auf dem Bildschirm (Pixelbasiert). Textur und Beleuchtung wird für jedes Pixel anhand der Lage zu den drei Eckpunkten des zugehörigen Dreiecks ermittelt

Beleuchtung

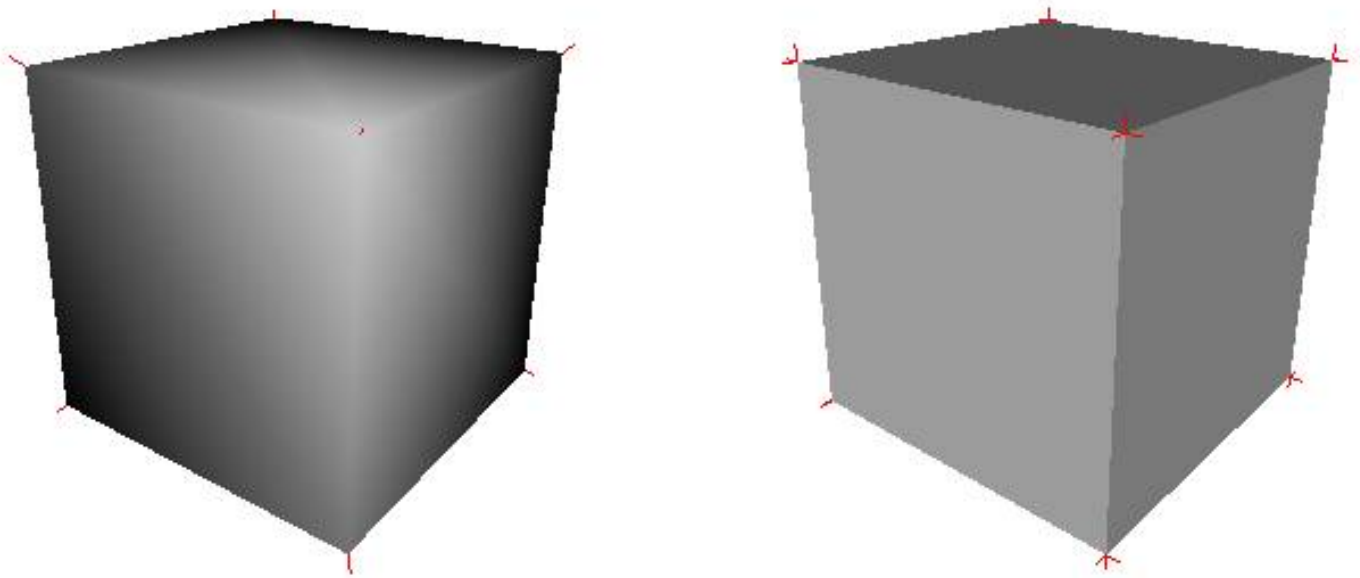


- Jeder Vertex hat einen Normalenvektor
- Normalenvektor steht i.d.R. senkrecht auf der Fläche, die der Vertex beschreibt
- Für jeden Vertex wird einzeln mit Hilfe des Normalenvektors der Lichteinfall berechnet
- **Flat-Shading**: Das gesamte Dreieck wird mit dem Mittelwert der 3 Farben gefüllt.

- **Gouraud Shading**: Die Füllfarbe der Dreiecksfläche wird zwischen den 3 Punkten interpoliert. Die Geometrie des Dreiecks bleibt unverändert “platt”, es entsteht durch die Farbverteilung lediglich die Illusion einer gewölbten Fläche.

Normalenvektor: Anwendungsfälle

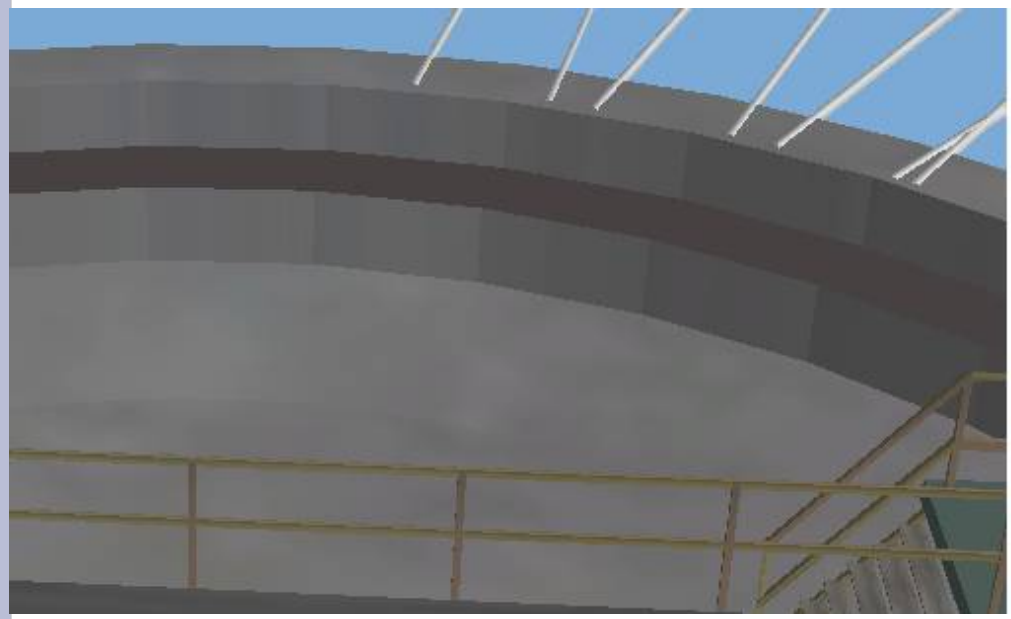
- Scharfe Ecken gewünscht: Mehrfache Vertices mit eigenen Normalenvektoren nötig
- Links: Falsche Beleuchtung, da nur ein Vertex pro Ecke
- Rechts: In jeder Ecke drei Vertices (einer für jede Fläche) Ergebnis ist ein korrekt beleuchtetes Modell

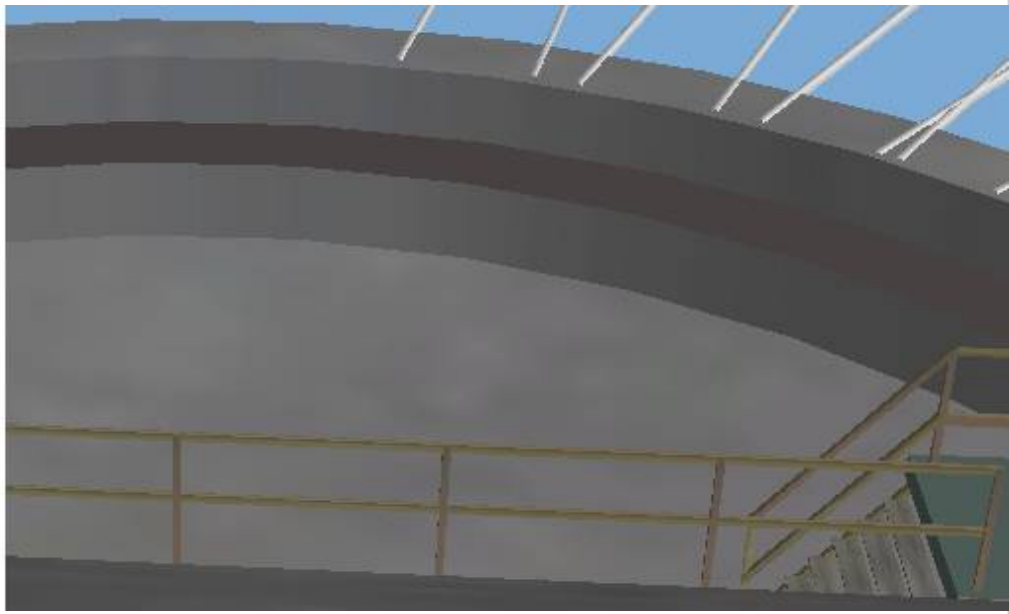


- **Ganz wesentlicher Punkt für realistisches Aussehen!**

Normalenvektor: Anwendungsfälle

- Runder Eindruck gewünscht: Nur ein Vertex pro Ecke – spart als Nebeneffekt etwas Speicher und Rechenzeit
- Links: Falsche Beleuchtung, da Vertices doppelt erzeugt wurden (Grubenrand wirkt eckig)
- Rechts: Mit der Funktion „Mesh-Optimieren“ wurden die Vertices zusammengeführt, die Grubeninnenseite wirkt rund. Der Übergang in die Waagerechte (korrekt) eckig

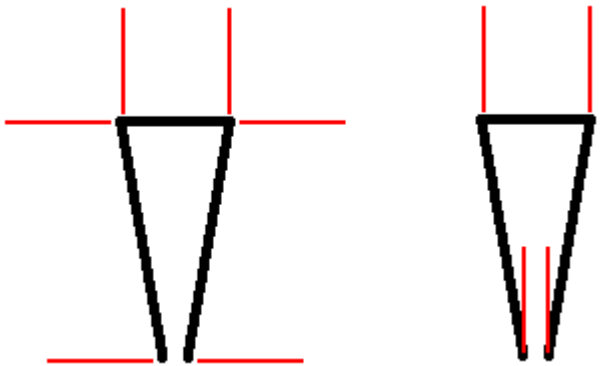




Normalenvektor: Anwendungsfälle

- Baum aus „Sechseck“: Der Fächer soll seine Struktur eigentlich nicht offenbaren – also alle Flächen müssen gleich hell beleuchtet werden.
  Lösung: Alle Normalenvektoren nach oben ausrichten. Baum wird dann komplett gleichmäßig mit der flacher einfallenden Sonne dunkler

- Schiene: Korrekt betrachtet 6 Vertices nötig (links)
  Trick spart 1/3 der Vertices, und fällt bei so niedrigen Objekten wie Schienen bei geschickter Textur nicht auf: Alle Normalenvektoren nach oben ausrichten (rechts)



Texturgrundlagen

- Textur ist externe Bitmap-Datei
- dds-Format dringend zu empfehlen, aber auch tga, bmp, png, dib, jpg technisch möglich
- Nur quadratische Texturen mit 2^N Pixel Kantenlänge (2x2, 4x4, 8x8, 16x16 usw.)
- Zuordnung der Textur zu den Dreiecken des 3D-Modells über Texturkoordinaten u, v
- Gleiche Bildbereiche nur einmal auf der Textur: Textur ist also oft völlig anders aufgeteilt als das Modell
- Transparenz wird über den Alpha-Kanal der Textur geregelt
- Grundlegend anderes Verhalten zwischen Halb- und Volltransparenz
- Diverse Artefakte und Probleme werden auf den folgenden Folien erläutert

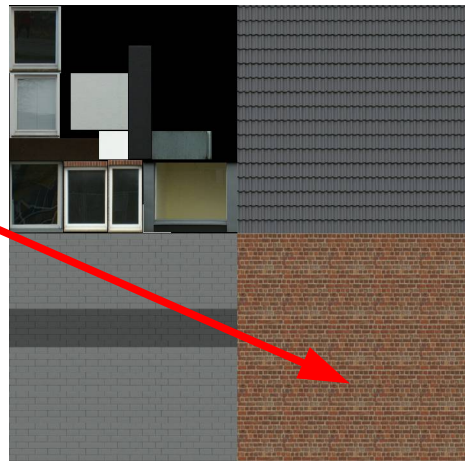
Texturkonzept und -größe

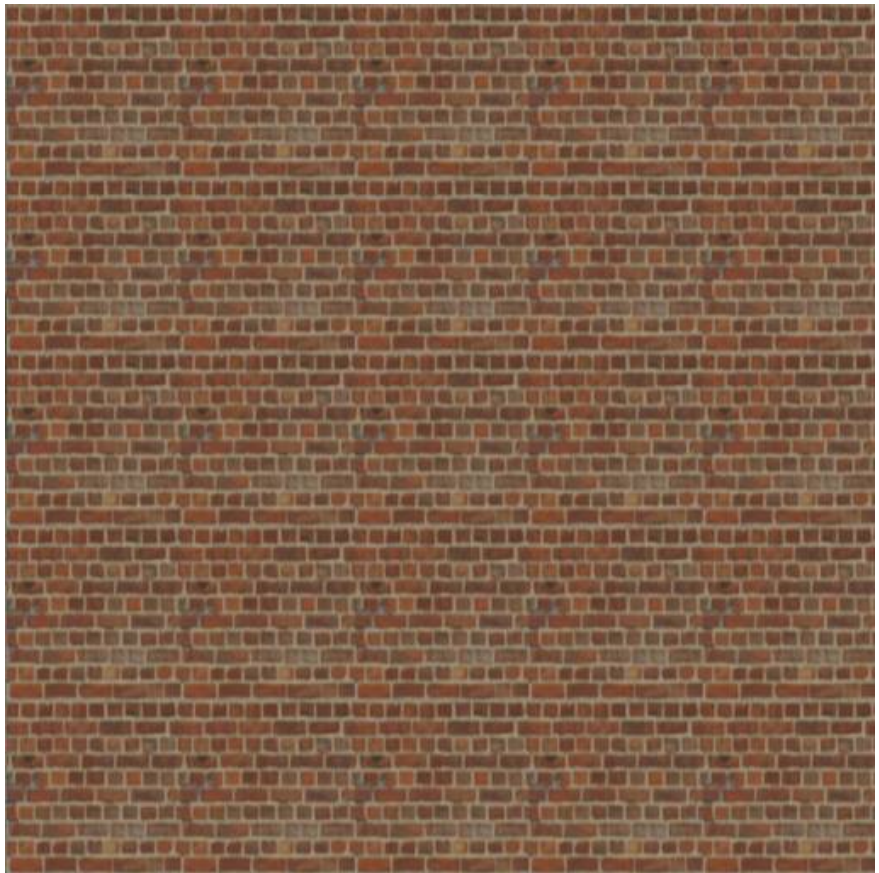
- Alle Texturbereiche mit etwa gleicher Skalierung benutzen, fein strukturierte Bereiche aber ggf. etwas größer, gleichmäßig strukturierte Bereiche etwas kleiner
- Im Normalfall reicht eine Texturgröße kleiner als durch Mipmaptest offenbart, ohne daß für das Auge wahrnehmbare Unterschiede entstehen
- Unbedingt unter realistischen Randbedingungen testen (Aus dem Gefühl heraus extrem schwer zu beurteilen)!
- Textur gut ausnutzen, ggf. können sich mehrere Objekte eine Textur teilen, wenn sonst keine sinnvolle Aufteilung möglich
- Kleinere Modellvarianten auf einer Textur unterbringen
- Wenn Platz knapp nicht reicht, separate Mesh-Subsets mit eigener Textur ausrüsten (z.B. bei Animationen, die sowieso separate Subsets benötigen)

Texturen kacheln

- Texturen können endlos gekachelt werden
- Aber nur die ganze Textur ist kachelbar
- Vorteil: Eine kleine Textur kann mit wenigen Polygonen eine große Fläche belegen
- Nachteil: Es entstehen Wiederholungsmuster und für andere Oberflächentypen im Objekt ist 2. Textur nötig
- Bei Häuserdächern o.ä. kann daher eine „Kachelung per Polygon“ sinnvoller sein

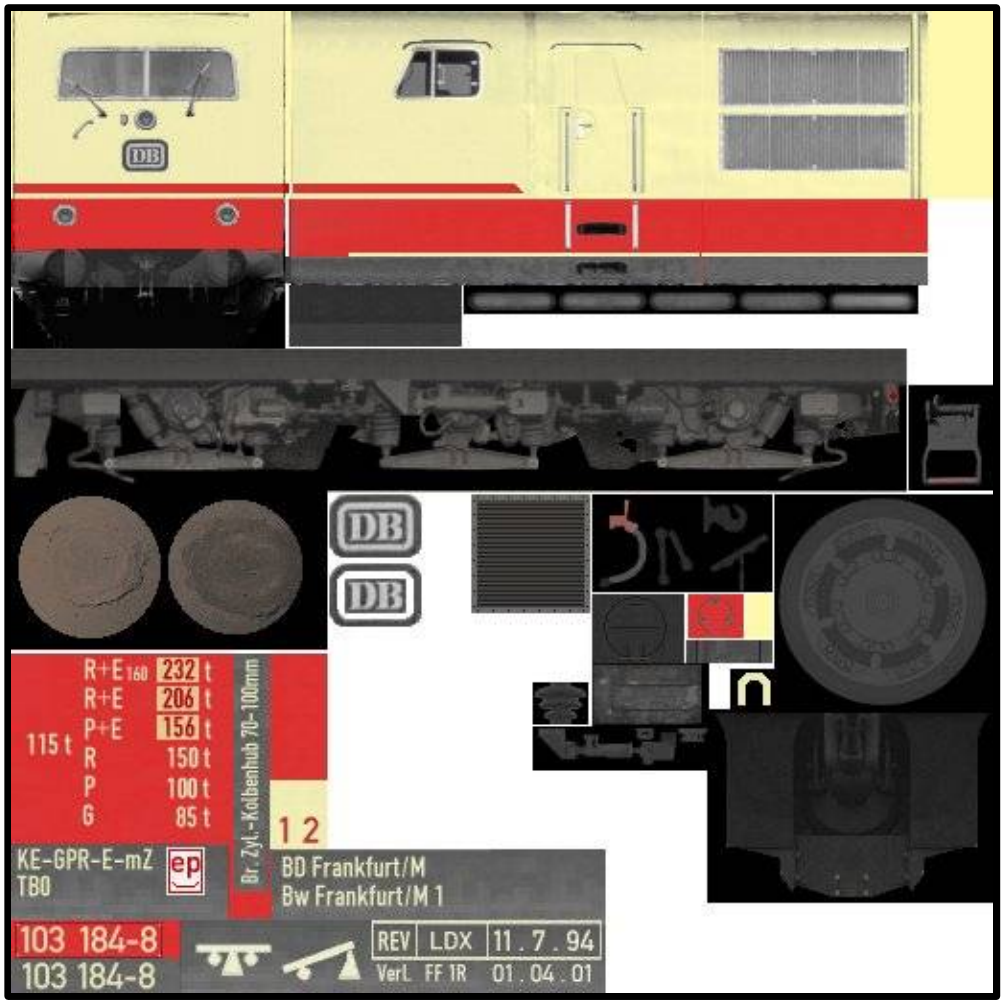
Ziegelkachelung nur über Polygone möglich





Beispiel 103 von Stephen Graham

- Gutes Beispiel: BR 103
- Front, Seite, Tür, Lüfter je einmal auf der Textur
- Textur gut ausgenutzt
- Beschriftungen in etwas größerer Skalierung
- Platz für Lok- Varianten auf der Textur
- DB-Keks auf der Front sollte noch entfernt werden
- Bei knappen Platz auch halbe Front ausreichend



Beschriftungen

- Beschriftungen können auf eigenes Polygon gesetzt werden (Austauschbarkeit und höhere Skalierung möglich)
- Polygon muß nicht eingeschnitten werden: Wenn z-Bias auf -1 gesetzt wird, erscheint die Textur über der auf gleicher Höhe liegenden Grundfläche (aber eigenes Mesh-Subset nötig)

Volltransparenz

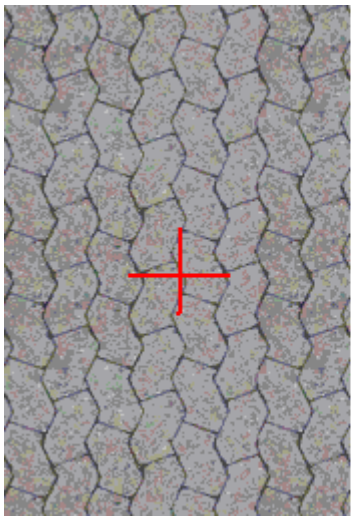
- Volltransparenz ist der einfachere Fall
- Pixel sind entweder komplett sichtbar oder komplett unsichtbar
- Unsichtbare Pixel werden im Alpha-Kanal der Textur definiert
- DXT1-Format mit 1 bit Transparenz reicht dafür schon aus
- Mesh-Einstellung auf „Volltransparenz“
- Mehr ist für die Funktion nicht zu beachten
- Auch transparente Flächen kosten Rechenaufwand
- Daher ist statt großer transparenter Flächen ein zusätzliches Polygon oft sinnvoller

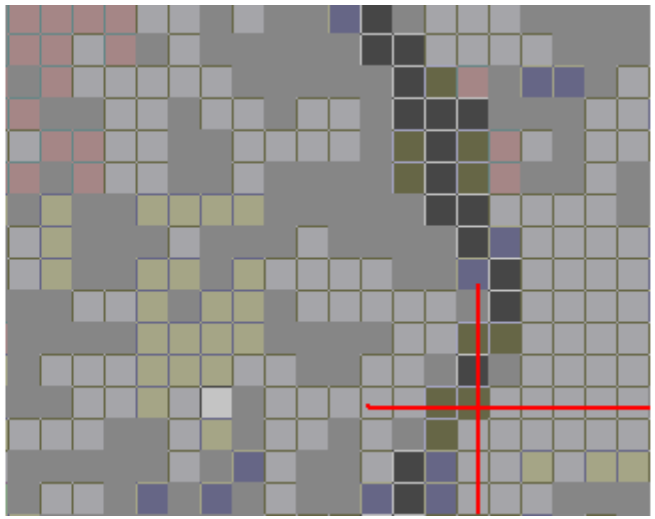
Halbtransparenz

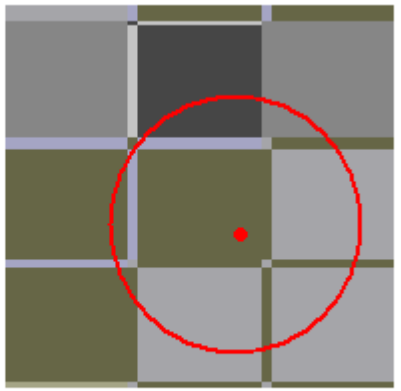
- Halbtransparenz erlaubt weichere Transparenz-Übergänge und Effekte wie halbdurchsichtige Scheiben
- Transparenz wird stufenlos (8 bit) im Alpha-Kanal der Textur definiert (DXT3-Format)
- Mesh-Einstellung auf „Halbtransparenz“
- Das aktuell gezeichnete Dreieck wird mit den bereits an der Stelle vorhandenen Bildschirmpixeln gemischt
- Rechenaufwendig!!
- Die Reihenfolge beim Zeichnen ist wichtig für korrekte Darstellung
- Zusi sortiert Objekte automatisch von hinten nach vorne
- Innerhalb eines Objekts muß der Bastler Mesh-Subsets mit Halbtransparenz ans Ende der Subsets schieben (Beispiel 401-Lampen)

Von der Textur auf den Schirm

- Analog zur Lichtberechnung werden für jedes Dreieck die Texturkoordinaten der Eckpunkte betrachtet und dazwischen für jedes Pixel auf dem Bildschirm interpoliert
- Es wird also nicht wie in einem Fotoprogramm das Bitmap auf das Polygon gestreckt/gestaucht, sondern „rückwärts“ für jedes Pixel auf dem Bildschirm der zugehörige Farbwert aus der Textur ermittelt
- Grundlegend für das Verständnis diverser Artefakte!
- Beim Ermitteln des Farbwerts wird nur im Bereich eines Texturpixels gefiltert (wie exakt regeln die Texturfilter)







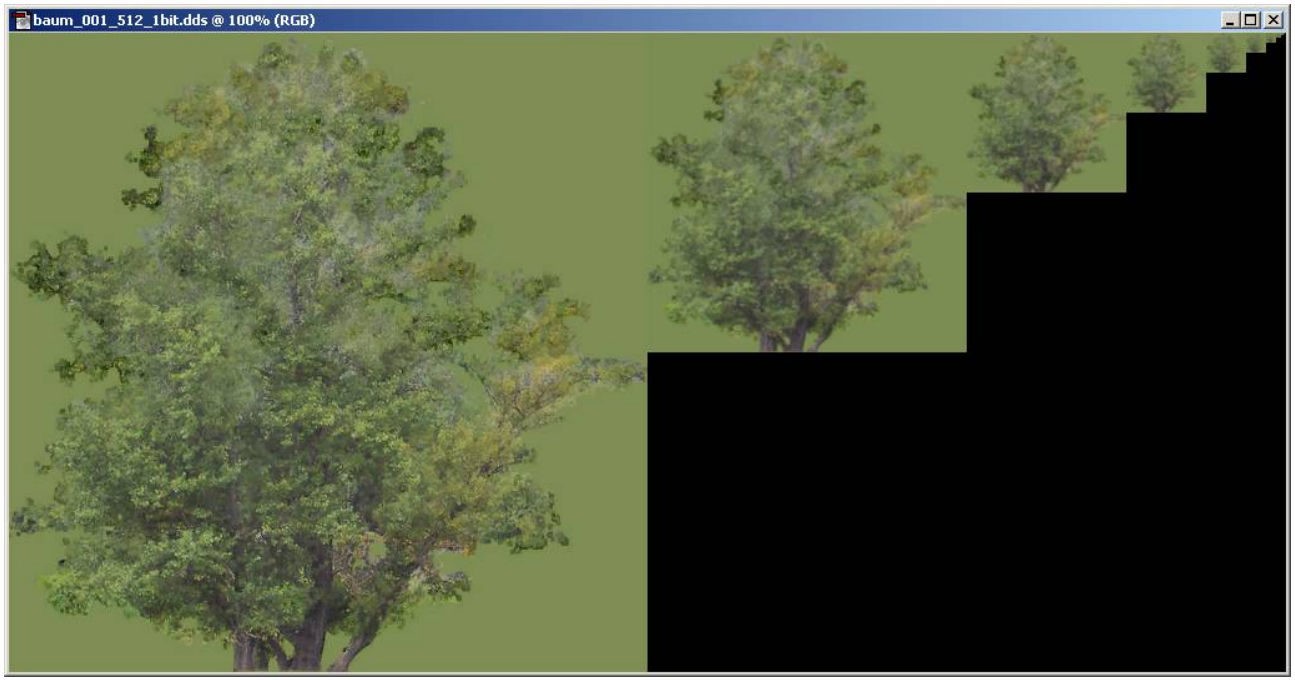
Artefakt: Aliasing

- Wenn das gerade gezeigte Straßenpflaster recht klein auf dem Bildschirm erscheint, werden nur einzelne Pixel der Textur für das Füllen des Bildschirmdreiecks abgegriffen – welche das genau sind, hängt von der aktuellen Perspektive ab
- Schon bei minimal veränderter Perspektive werden andere Pixel abgegriffen
- Hier und da wird immer mal ein Pixel gerade zufällig eine dunkle Fuge treffen
- Pixel wechseln also zufällig zwischen hell und dunkel: Wildes Flimmern der Oberflächen
- Zu **große** Textur: I.d.R. Flimmern (!)
- Zu kleine Textur: I.d.R. unscharfe Darstellung

- Lösung des Konflikts: Mipmaps
  (Beispiel schottertest1.ls3)

Mipmaps

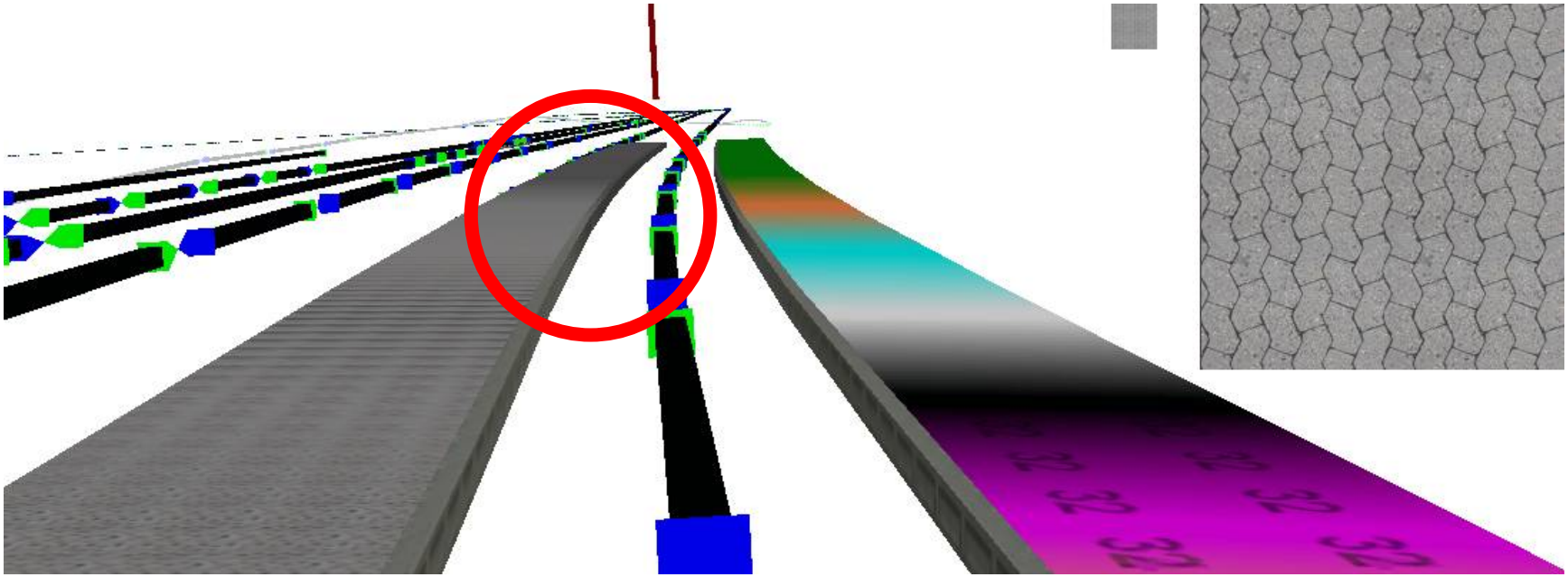
- Textur wird im Ausgangsformat und gestuften kleineren Formaten benutzt



- Beim Zeichnen wird für jedes einzelne Bildschirmpixel die gerade passende Texturgröße automatisch bestimmt
- Kaum Flimmern, da feine Strukturen wie die Fugen bereits in den Mipmapstufen gefiltert werden (schottertest2.ls3)
- Trotzdem Artefakte möglich, dazu mehr in folgenden Folien

Mipmap-Artefakte

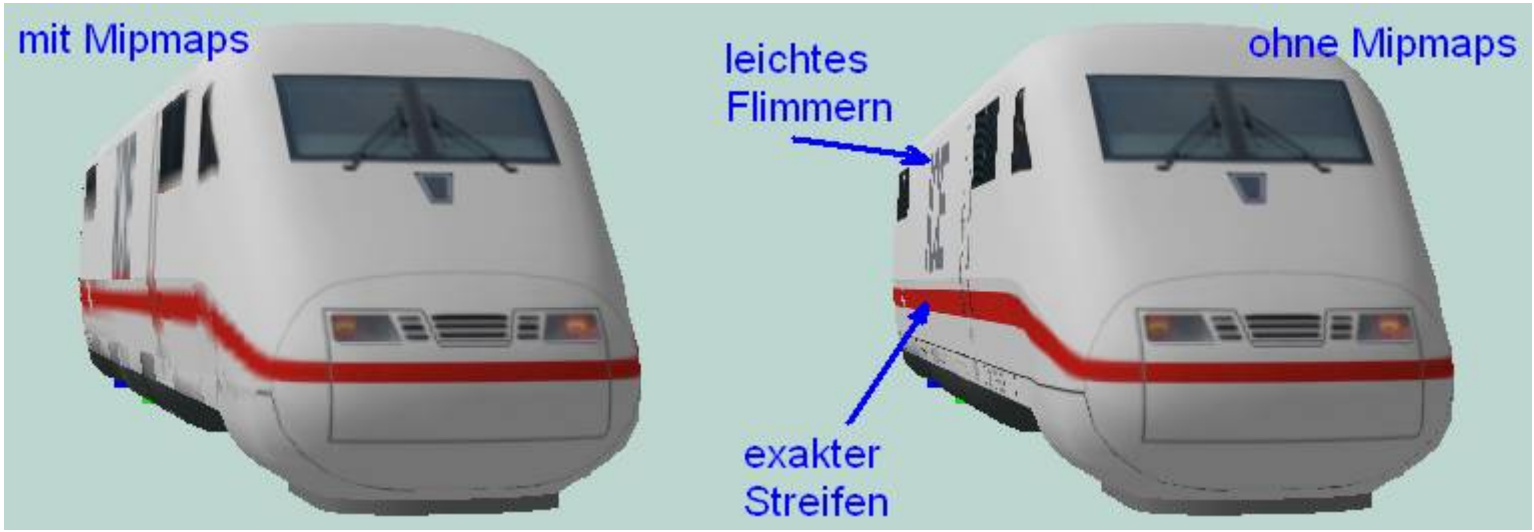
- Kleinere Mipmaps können bei automatischer Generierung ungünstige Farbwerte erhalten
- Beispiel: Bahnsteigpflaster ganz am Ende (1x1-Mipmap)



- Lösung: Die problematische Mipmapstufe manuell erzeugen (z.B. Photoshop oder Texturetool), evtl. auch Entfall der letzten Stufe?

Mipmap-Artefakte: Feine Linien

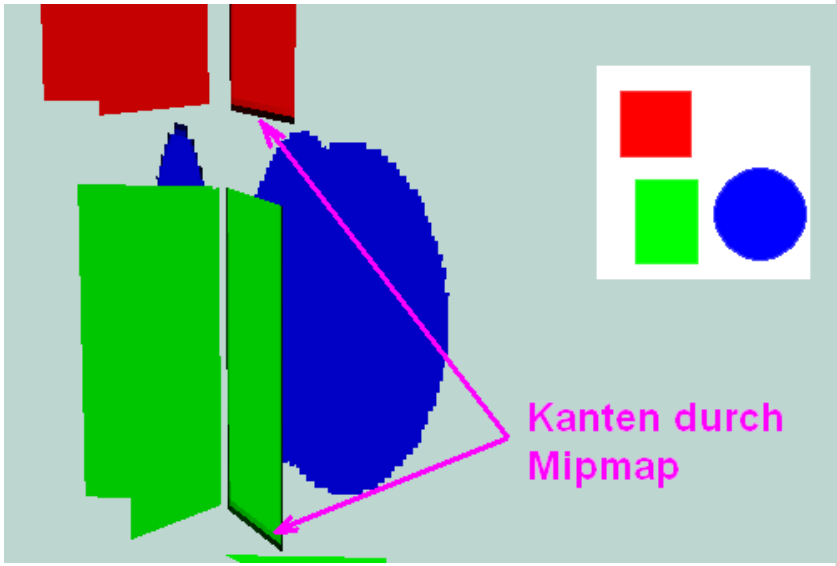
- Feine Linien verschwinden oder verwischen bei schrägem Blickwinkel (Beispiel linientest.ls3)
- Mipmap manuell anpassen und z.B. feine Linien in kleinen Mipmapstufen ganz entfernen
- Evtl. ganz ohne Mipmaps arbeiten, wenn sich die Flimmereffekte dank recht gleichmäßiger Oberflächenstruktur in Grenzen halten (bei manchen Fahrzeugen vorstellbar)



- An der Trennkannte eine Polygonkante einbauen, da diese immer exakte Farbtrennung bewirkt (z.B. roter Streifen bei „Lazarettlackierung“ der aktuellen IC)

Mipmap-Artefakte: Transparenzkante

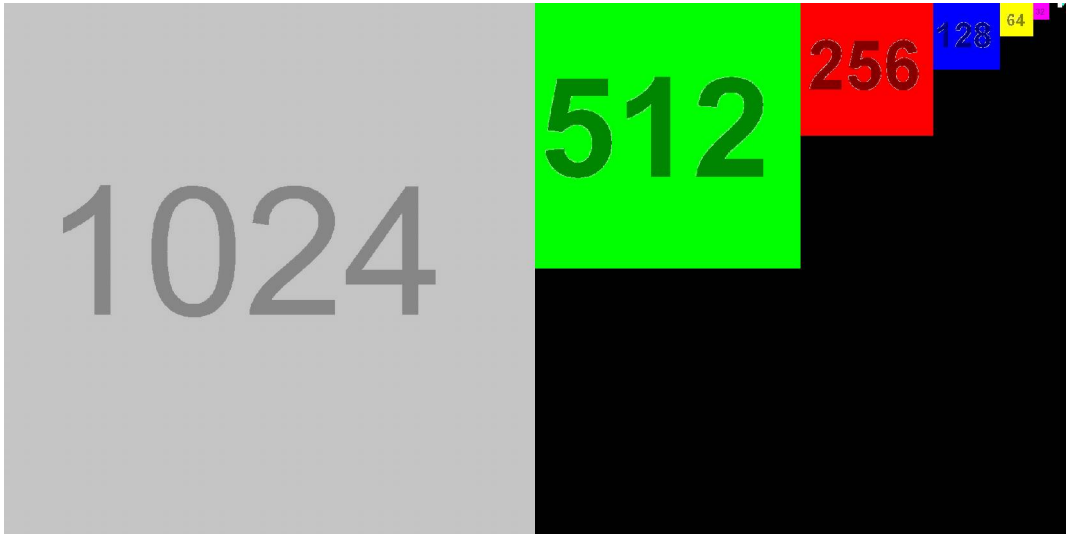
- Harte transparente Kanten werden ungenau, da der Transparenz-übergang „weichgefiltert“ wird
- Hier wird durch DXT1-Format die weiße Fläche in schwarz gewandelt -> schwarze Ränder

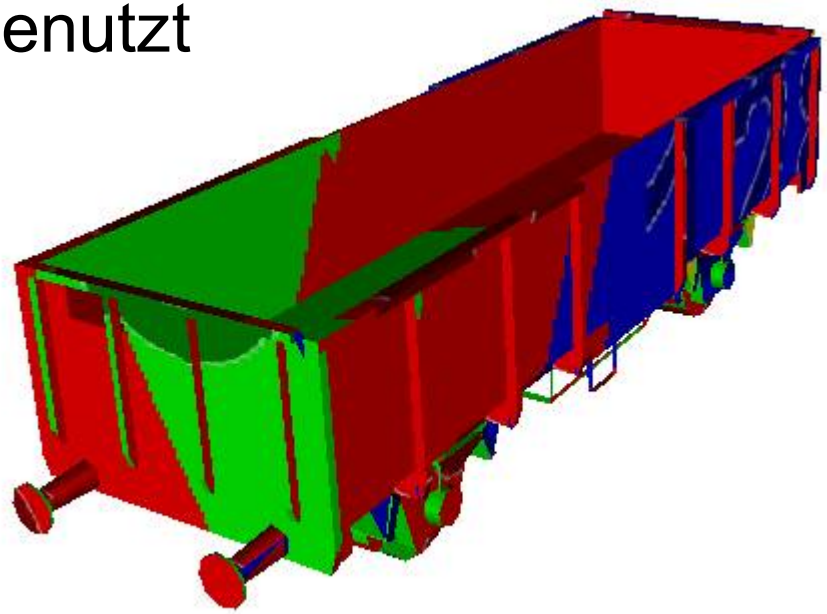


- Kann umgangen werden durch Färben der Transparenzbereiche in der Randfarbe (Hinweis: geht nicht mit dds-DXT1-Format, DXT3 nötig)
- Hinweis: Derartige Ränder können auch durch die Texturfilter entstehen (D3DSAMP_MINFILTER und D3DSAMP_MAGFILTER im Editor)

Abfallprodukt Mipmap-Test

- Individuelle Mipmap-Farben offenbaren, welche Texturgröße DirectX tatsächlich zum Zeichnen benutzt



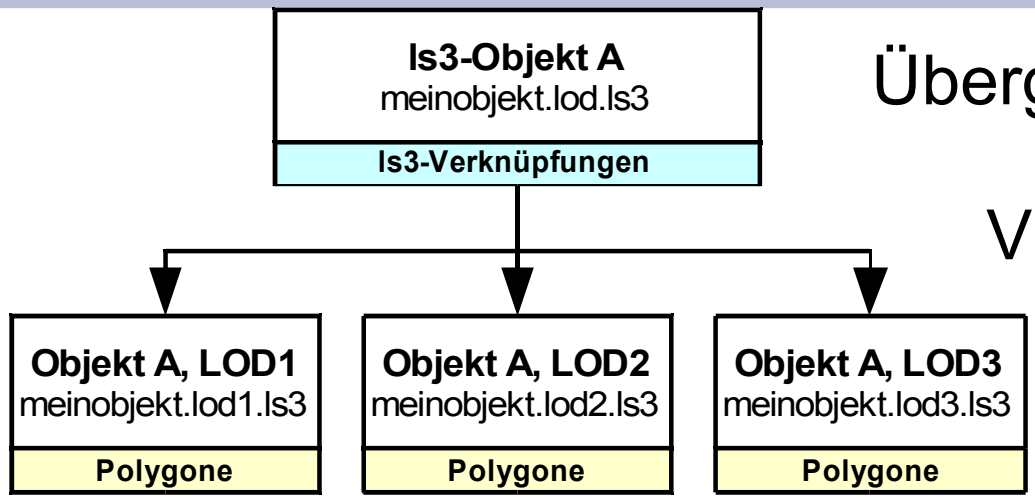


- Wichtiger Hinweis für Objektbauer auf sinnvolle Texturgrößen
- Scharfer oder weicher Mipmap-Übergang je nach Mipmap-Filter (Editor-Einstellung)
- Mipmap-Test mit einem Mausklick im 3D-Editor aktivierbar
- **Unbedingt testen**, aus dem Gefühl heraus ist das extrem schwer zu beurteilen!
- Nächstkleinere Texturgröße i.d.R. Ohne optische Einbußen

LOD – Level of detail

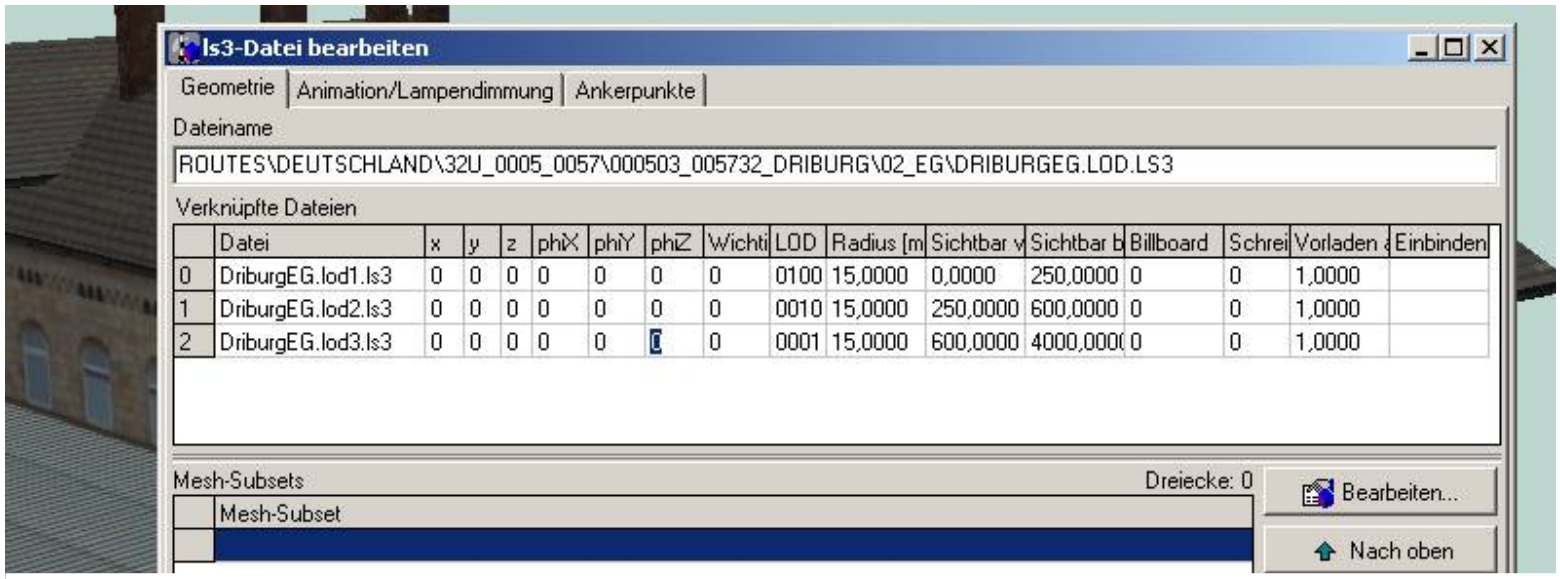
- Ein 3D-Objekt wird in mehreren Detailstufen erstellt
- Nur nahe Objekte werden als aufwendiges Modell dargestellt
- Entfernte Modelle sind einfache Kisten, evtl. auch ohne Textur
- Rechenaufwand sinkt drastisch
- Gerade bei entfernten Modellen ist ganz besonders auf performancefreundliche Bauweise zu achten, da davon deutlich mehr gleichzeitig sichtbar sind als von den nahen Modellen
- Drei LODs + ein optionaler Detaillevel ist Standard für Zusi
- Die drei Modelle müssen sinnvoll abgestuft werden
- Ggf. auch weniger LODs sinnvoll, z.B. bei einfachen Gebäuden oder Bäumen, die sowieso nur sehr einfache Polygonstruktur aufweisen
- 3D-Editor hat speziellen LOD-Test-Modus, in dem die LODs nebeneinaner dargestellt werden, um die Übergangsentfernungen festzulegen

LOD-Datei



Übergeordnete Datei enthält keine 3D-Daten, sondern nur Verknüpfungen zu den LODs

Jeder LOD ist eine eigene 3D-Datei



Der Streckenbauer baut nur die übergeordnete Datei ein. Der Rest (LOD-Sichtbarkeit usw.) wird innerhalb des Objekts geregelt

LOD-Test im 3D-Editor

- LOD-Test-Modus lädt alle LODs nebeneinander
- Mit dem Abstandsregler lassen sich Entfernungen einstellen
- Zur Bestimmung der Umschaltentfernungen: Niedriges LOD reicht aus, wenn kein allzu großer Unterschied mehr zum höheren erkennbar ist
- Realistische Bildschirmauflösung beachten

- Grobe Richtwerte: LOD0 bis ca. 50 m, LOD1 bis ca. 200m LOD2 bis ca. 500m
- **Unbedingt testen**, aus dem Gefühl heraus ist das extrem schwer zu beurteilen!

Nachtdarstellung

- Erster Ansatz: Eine zweite in Aufteilung und Größe identische Textur enthält die beleuchteten Fenster und ist sonst schwarz. Die Darstellung für das Meshsubset wird auf Nachtmodus gestellt, so daß die Fenster stets in der gleichen Helligkeit dargestellt werden (Analog Zusi 2-Führerstandskonzept

- Zweiter Ansatz: Die beleuchteten Fenster werden als eigenes Mesh-Subset ausgeschnitten und mit einer passenden Nachtfarbe versehen (analog Zusi 2-Objektbau). Besonders dann sinnvoll, wenn nur kleine Flächenanteile beleuchtet werden sollen, da die 2. Textur entfällt

Animationen

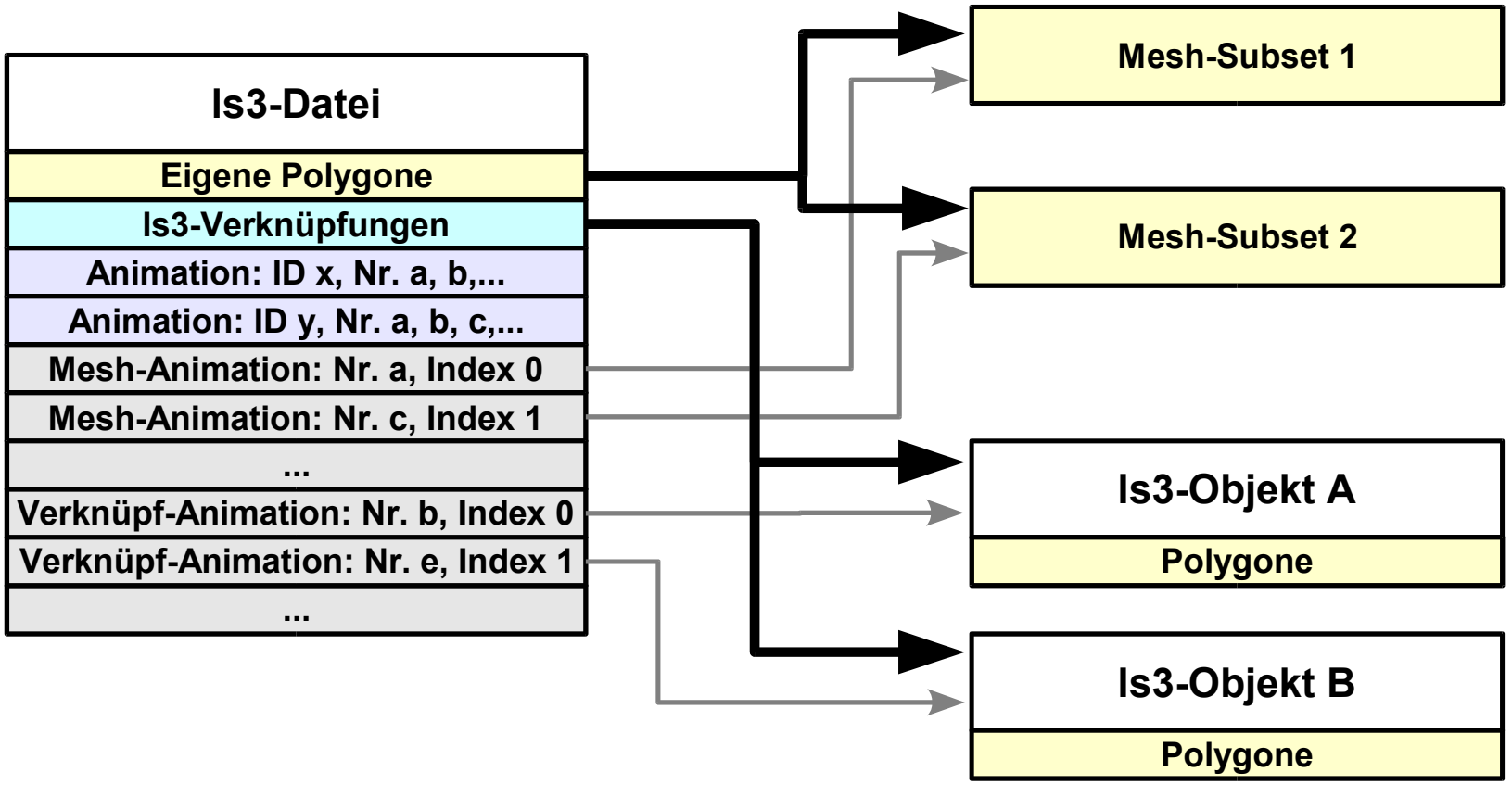
- Zusi-eigenes Konzept in Anlehnung an .x-Dateien
- Es können einzelne Mesh-Subsets oder auch verknüpfte Dateien verschoben und/oder um Achsen rotiert werden
- Dazu wird ein Verschiebungsvektor oder eine Drehachse + Winkel definiert. Beliebig viele Stützpunke möglich – dazwischen wird interpoliert

- Hierarchien möglich
- LOD-übergreifende Animation möglich
- Eingabe unter ls3-Eigenschaften, Geometrie der Bewegung ist also integraler Bestandteil der ls3-Datei
- Steuerung/Logik erfolgt über Zusi
- Testen direkt im Editor möglich

- Lampendimmung für das Umschalten von Lichtsignalen nach dem gleichen Schema möglich, auch wenn es technisch ein völlig anderer Vorgang ist

Animation im Detail

- Ein Bewegungsablauf muß mehreren Animationen zugeordnet werden können (oberer Signalflügel Hp1 + Hp2)
- Indizierung muß LOD-unabhängig, also hierarchierübergreifend eindeutig sein



Vorschau: Schatten

- Vorgesehen für Zusi 3: Shadow-Maps
- Szene wird zusätzlich aus Sicht der Sonne gezeichnet und das Ergebnis in hoch aufgelöster Textur zwischengespeichert
- Diese Textur enthält jetzt eine 3D-Tiefeninformation, aus der sich ableiten läßt, ob ein Punkt von der Sonne beschienen wird
- Aus Betrachtersicht läßt sich dann über eine Transformation ermitteln, ob ein Pixel im Schatten oder in der Sonne liegt
- Aufwendig! Daher Anzahl der Polygone begrenzen: Polygone kennzeichnen, die Schatten werfen können und auf die Schatten fallen kann
- Vorteil: Transparente Objekte werfen korrekte Schatten
-
- (Demo)

3D-Editor

- Grundsätzliche Einstellung: Objekt- oder Streckenbaumodus
- Objektbaumodus entspricht etwa dem Gebäudeeditor
- Streckenbaumodus führt ggf. automatische Kachelungen durch
- Eine Kachel ist 1000x1000m groß und am UTM-Raster orientiert
- Vorteil: Fahrsimulator kann die Landschaft rund um den Betrachterpunkt gezielt laden
- Kacheln sind verknüpfte Dateien unterhalb der Landschafts-Hauptdatei
- Zuordnung von Objekten zu den Kacheln erfolgt automatisch beim Import
- 3D-Editor kann alle dargestellten Dateien bearbeiten, egal in welcher Hierarchie-Ebene sie sich befinden

Hierarchie der Landschaft