Semesterarbeit
Computer Grafik Animation

BLITZBASIC

Inhalt::

Was ist BlitzBasic?

Als ich das erste Mal mit dieser Programmierumgebung konfrontiert wurde, hat sich bestätigt was man intuitiv beim Hören des Namens erwartet: eine Basic ähnliche Programmiersprache. Jedoch verpackt in einer Entwicklungsumgebung, wie man sie aus Zeiten von Pascal und Co. gewohnt war. Dies beinhaltet sowohl die von daher gewohnte Programmieroberfläche, als auch die sehr bequeme Kontexthilfe und den Build-in-Compiler der auch ohne das Erzeugen von Executables den erstellten Quelltext innerhalb der Entwicklungsumgebung starten kann. Blitzbasic bezeichnet dieses als IDE, Integrated Development Environment. Dies ist das Herzstück und das Hauptfeature, welches die Demo von Blitzbasic überhaupt in dieser Weise nutzbar macht. Der Quellcode lässt sich in der Demo leider nicht in ausführbare Dateien umsetzen, sehr wohl aber innerhalb der Programmierumgebung testen.
Trotz alledem ist Blitzbasic nur begrenzt eine Umgebung für die Erstellung professioneller Arbeitsprogramme. Um so mehr ist es eine Entwicklungsumgebung für professionelle Spieleprogrammierung, denn die eigentliche 3D-Engine ist bei Blitzbasic bereits integriert. So muss der Programmierer nicht wie bei C++ oder Visual C sich erst um eine Schnittstellenimplementierung, wie OpenGL / Direct 3D, kümmern und die eigene Grafikengine darauf abstimmen, sondern kann direkt auf Funktionen der Engine zugreifen. Das mag auch seine Nachteile haben in Bezug auf Features, die diese Engine möglicherweise nicht deckt. Jedoch möchte ich hier auch nicht BlitzBasic als ultimatives Grafikwunder hervorheben, sondern als leicht verständliche, gut dokumentierte und vor allem schnelle Entwicklungsumgebung.
So kann Blitzbasic zum Beispiel mit überragenden Frameraten überzeugen, selbst auf Low-End Systemen wie Pentium II 300 mit Voodoo3 Grafikkarte und 128 MB Arbeitsspeicher sind Frameraten mit bis zu 30fps bei Auflösungen bis zu 800x600x16 möglich. Vergleicht man das mit heutigen Spielen auf dem Markt so deckt Blitzbasic ein relativ gutes Leistungsspektrum ab.
Zum Beispiel kann die Blitz3D-Engine durch eine gute Optimierung im Terrain durch Level of Detail diese Performance herauskitzeln, wie ich zu einem späteren Zeitpunkt in dieser Ausarbeitung noch erläutern werde.

Was kann Blitzbasic?

Ich möchte diesen Abschnitt der Semesterarbeit dafür verwenden, um einige der Features von BlitzBasic aufzuzeigen. Dabei werde ich nicht großartig ins Detail gehen sondern Features die des Details bedürfen in späteren Abschnitten gesondert beleuchten.

Die Features von Blitzbasic lassen sich im Großen und Ganzen in folgende Kategorien unterteilen:

- 2D Grafik
- 3D Grafik
- Audio
- Netzwerk
- Sprache
- File IO

Das Augenmerk dieser Semesterarbeit liegt hierbei allerdings auf den 2D/3D Grafik Features. Daher werde ich die anderen Kategorien nur kurz anschneiden.

Wenden wir uns zunächst der Grafik unter BlitzBasic/Blitz3D zu.
Die Grafikmodi unter BlitzBasic richten sich generell nach der Grafikkarte des Nutzers, daher ist es an sich immer ratsam, bei der Programmierung auf Grafikmodi zurückzugreifen, die eine Vielzahl von Grafikkarten unterstützt, oder die Funktionen zu nutzen die Blitzbasic zur

Verfügung stellt, um die Modi zu ermitteln und z.B. den Nutzern die Wahl zu überlassen. Der Einfachheit halber sind alle Beispiele für diese Semesterarbeit in 800x600 Bildpunkten mit 16Bit Farbtiefe und Vollbild erstellt. Bis auf die BSP-Demo sind alle auch unter der Demo von Blitz3D lauffähig.
Funktionen, die bei der Lokalisierung der Grafikmodi helfen, sind:

- *CountGfxModes()*
- *GfxModeHeight()*
- *GfxModeWidth()*
- *GfxModeDepth()*

Sowie

- *GfxModeExist()*

CountGfxModes() liefert hierbei einen Index zurück, welcher der Gesamtzahl aller verfügbaren Grafikmodi entspricht. Alle anderen Funktionen nutzen einen Wert innerhalb dieses Index, um für den entsprechenden Index die zugehörigen Werte zurückzuliefern. So kann man sich z.B. einfach mittels Zählschleife alle verfügbaren Grafikmodi ausgeben lassen. Bei der eigentlichen Grafikinitialisierung kommen dann, je nachdem ob man im 2D oder 3D Bereich arbeiten möchte,
Graphics *int width, int height, int depth, boolean fullscreen* bzw.
Graphics3D *int width, int height, int depth, boolean fullscreen*
zum tragen. Beide initialisieren die Grafikengine von Blitz mit dem jeweiligen XY oder XYZ Koordinatensystem.

2D Grafik:

Für die 2D-Grafik stehen diverse einfache Befehle zur Verfügung, um schnelle Ausgaben von Rechtecken, Blöcken, Linien, Ellipsen und Bildern zu gewährleisten; ähnlich wie z.B. in der Graphics-Umgebung von Turbo Pascal. Hat man jedoch damals bei Pascal z.B. bei Spritebewegungen versucht, kleinstmögliche Flächen um die Sprites zu aktualisieren um einen Flimmereffekt zu vermeiden, kann man hier sehr bequem auf das Double Buffering-Verfahren zurückgreifen, um einen flüssigeren, flimmerfreien Bildaufbau zu gewährleisten. Bemerken möchte ich hierbei die Einfachheit dieser Funktion, die durch zwei simple Codezeilen umsetzbar ist.
Mittels *SetBuffer BackBuffer()* wird ein zweiter Speicherbereich für die Anzeige geschaffen, so dass nun die Möglichkeit besteht, zwischen dem Frontbuffer den es ja standardmäßig immer gibt und dem Backbuffer hin und her zu schalten.
Dieses Schalten passiert mit dem Befehl *Flip.* Alle Manipulationen an Objekten, Sprites, Texten etc. können damit bereits auf der hinteren nichtsichtbaren Ebene gerendert werden, noch während der andere Buffer ausgelesen wird. Durch diese Funktion können besonders weiche Animationen geschaffen werden und das Flimmern des Bildschirmes beim „Überzeichnen“ wird dadurch unterdrückt. *(Siehe auch 2D_DEMO.bb / 2D_DEMO.exe)*

Weitere Besonderheiten des 2D Bereichs sind sowohl die Unterstützung weitverbreiteter Bildformate wie Bitmap, JPG, PNG, GIF, TGA etc. als auch dessen Nutzung für Bildsequenzen. Diese „animierten“ Bilder können Bildsequenzen in einem Bild mit einer eigenen Höhe und Breite enthalten, auf diese Weise lassen sich sehr einfach benutzerdefinierte Bitmapfonts erstellen aber auch z.B. Spriteanimationen. Die Blitzbasic-Dokumentation spricht hierbei gerne von Multiframe Animationen.

3D Grafik:

Kommen wir zu den Besonderheiten der 3D-Grafik. Neben den Standardbefehlen zum Erstellen einfacher geometrischer Strukturen bietet Blitz3D noch diverse Importmöglichkeiten von 3D Strukturen wie 3DS, MD2 und DirectX-Modelle. Bei 3DS-Modellen kann, falls diese texturiert sind und die Texturen ordnungsgemäß beiliegen, diese gleich mit angezeigt werden ohne sich explizit bei Blitz3D nochmals darum kümmern zu müssen. Wo wir schon bei den

Texturen sind: Blitz3D bietet auch diverse Texturfeatures wie zum Beispiel Animated Textures, bei denen die Textur zur Laufzeit frameweise geändert werden kann. Multitexturing, Spheral Environment Maps, Mipmapping und Texturenwechsel zur Laufzeit sind auch keine Fremdworte für Blitz3D und zählen mit zu den zahlreichen Texturfeatures, die Blitz3D zu bieten hat.
Texturen können maskiert oder mittels Alphakanal in der Bildinformation transparent gemacht werden aber auch mittels Alphakanal, Multiplikation oder Addition überblendet werden. Ein weiteres sehr interessantes Feature hierbei ist noch, dass Texturen sich auch zur Laufzeit erzeugen lassen.
Was auch sehr interessant ist, ist die Tatsache das Blitz3D mehre Kameras unterstützt. Dadurch kann man den bekannten Splitscreen erzeugen sowie Bild-in-Bild-Darstellungen.
Weitere Features sind Render Tweening für weichere Grafiken bei jeder beliebigen Framerate, ein Dynamisches Level of Detail Terrain, auf das ich später noch detaillierter eingehen werde, eine schnelle Kollisionsabfrage, Vertex Coloring, Specular Highlights sowie Beleuchtungsmodelle wie Spot-, Punkt-, ambientes und direktes Licht.
Auch nennenswert ist die Eigenschaft während- der Laufzeit dynamisch 3D Modelle vertexorientiert deformieren zu können.
Was das Handhaben von 3D-Objekten bei Blitz3D wesentlich vereinfacht ist das Entity-(Einheits)-System. Hierbei wird jedes 3D-Objekt als Entity deklariert, wodurch es nicht typenspezifischer Funktionen bedarf sondern nur einer Funktion für eine jeweilige Aktion. Zum Beispiel das Drehen eines Objektes relativ zu seiner momentanen Ausrichtung, sei es eine Kamera, eine Lichtquelle oder ein 3D-Objekt wird mittels *TurnEntity entity, float x, float y, float z* realisiert, wobei in diesem Beispiel *entity* der Bezeichner des Objektes ist, zum Beispiel: *camera=CreateCamera().*
Um die Kamera also +1 Grad um die Z-Achse zu drehen sähe der Befehl wie folgt aus: *TurnEntity camera,0,0,1*.
Ziemlich einfach wenn man bedenkt, was es an sich an programmiertechnischem Aufwand bedeuten würde, diese Rotation um die Z Achse manuell in eine Funktion zu schreiben.
So gibt es für so ziemlich jede Manipulation eines Objektes eine Entity-Funktion, was das Arbeiten mit Blitz3D, als auch das Handhaben von Objekten im 3D-Raum wesentlich vereinfacht.

Audio:

Da die Audiofeatures nicht Teil dieser Semesterarbeit sind, werde ich sie nur der Vollständigkeit halber erwähnen. So bietet Blitzbasic die Möglichkeiten, Lautstärke, Balance und Frequenzen von Soundkanälen zu beeinflussen, sie zu unterbrechen und sogar an der unterbrochenen Stelle fortzusetzen, z.B. für Timingabgleich in Animationen, wodurch man nicht viele Audiodateien benutzen muss, sondern in einer Sequenz diese anhalten und an gegebener Stelle fortsetzen kann.
Auch unterstützt Blitzbasic gängige Formate wie .wav, .mp3 oder auch .x3m. In Blitz3D sind des weiteren auch Möglichkeiten gegeben, 3D-Sound zu erzeugen und somit grob die hörbare Position zu beeinflussen, um die Soundquelle besser orten zu können oder mit sichtbaren Ereignissen abzustimmen.

Netzwerk:

Ein weiterer Trumpf den Blitzbasic aufzubieten hat sind die Netzwerkfunktionen. Dadurch ist es möglich Spiele selbst internetfähig zu machen und Multiplayerfunktionen zu integrieren. Hierbei unterstützt Blitzbasic die Client/Server Architektur, wobei es auf das durch das Internet weit verbreitete TCP/IP Protokoll zurückgreift. Ein weiterer Punkt für einfache Multiplayerverwendung ist die Unterstützung der Microsoft DirectPlay Routinen.

Sprache:

Unter Sprachfeatures sind keine Funktionen für zum Beispiel Multilanguagesupport zu verstehen, sondern eher Features, welche die eigentliche Programmiersprache Blitzbasic unterstreichen. So sind, wie in vielen heutigen Programmiersprachen üblich, auch hier eigene Datentypen definierbar. Ähnlich den Recordsets anderer Sprachen. Ebenso gibt es eine Unterscheidung in lokale und globale Variablen mit derselben Gültigkeitsdauer wie in anderen Programmiersprachen auch. Zudem lassen sich explizit Variablen aus dem Speicher zu jeder Zeit freigeben. Für die Programmlogik bietet Blitzbasic Algorithmen wie Repeat/Until-Schleifen, For/Next-Schleife (äquivalent zur for..to Schleifen anderer Programmiersprachen), While/Wend-Schleifen (äquivalent zu kopfgesteuerten while - Schleifen) sowie For/Each-Schleifen. Letzterer Schleifentyp ist eine wirkliche Bereicherung, wenn man mehrere Objekte eines Typen hat und dadurch eine große Zahl von Objekten updaten möchte, wobei man dadurch explizit sagen kann: „*für jedes Objekt eines bestimmten Typs tu folgendes...*“.
Zum Vergleich kann man bei Blitzbasic auch neben den IF-Abfragen ein Select/case-Konstrukt benutzen um verschiedene Zustände einer Variable/Objektes abzudecken.
Wie bei jeder der mir bekannten Programmiersprachen lassen sich auch bei Blitzbasic eigene Funktionen hinzufügen und mittels include auch Dateien importieren. Das alles macht Blitzbasic zu einer schon recht komplexen Sprache die dadurch viel Potential aus der Kreativität des Programmierers schlagen kann.

File IO:

Als letzte Rubrik dieser Vorstellung möchte ich noch kurz die Datei Ein- und Ausgabe anschneiden. Hier bietet Blitzbasic sowohl die Unterstützung sequenzieller als auch random-access-Dateizugriffe. Ebenso gibt es umfassende Kommandos zum Lesen, Schreiben und Ändern von Text und Binärdateien.

Weiter möchte ich hier nicht auf die Features von Blitzbasic eingehen. Falls für manche hier angesprochenen aber nicht weiter ausgeführten Funktionen näheres Interesse besteht, kann eine Onlineversion des in der IDE enthaltenen Hilfetextes mit Definitionen, Syntax, Erklärungen und gelegentlichen Codebeispielen in der deutschen Blitzbase downloaden. Die dort enthaltenen Hilfen sind hierbei ausführlicher und anschaulicher als es teilweise die enthaltene Hilfe in der Entwicklungsumgebung ist. Zudem ist auch alles in deutsch dort verfügbar. Die URL findet man im Anhang dieser Semesterarbeit.

Kommen wir zu den Hauptgebieten dieser Semesterarbeit, diese werden zum Teil die Animationseffekte von Texturen, Änderungen des Terrains zur Laufzeit sowie den Support von BSP-Dateien behandeln. Veranschaulicht wird das hier geschriebene dabei durch Codebeispiele und Programme. Da die Unterstützung von BSP-Dateien bislang noch nicht 100%ig funktionstüchtig ist wie ich beim Experimentieren von diversen BSP-Dateien der Q2- und Q3-Generation feststellen musste, wird dieses Thema hier eher eine theoretische Abhandlung. Falls Resultate während dieser Bearbeitungszeit noch vorliegen, werden sie jedoch angehangen.

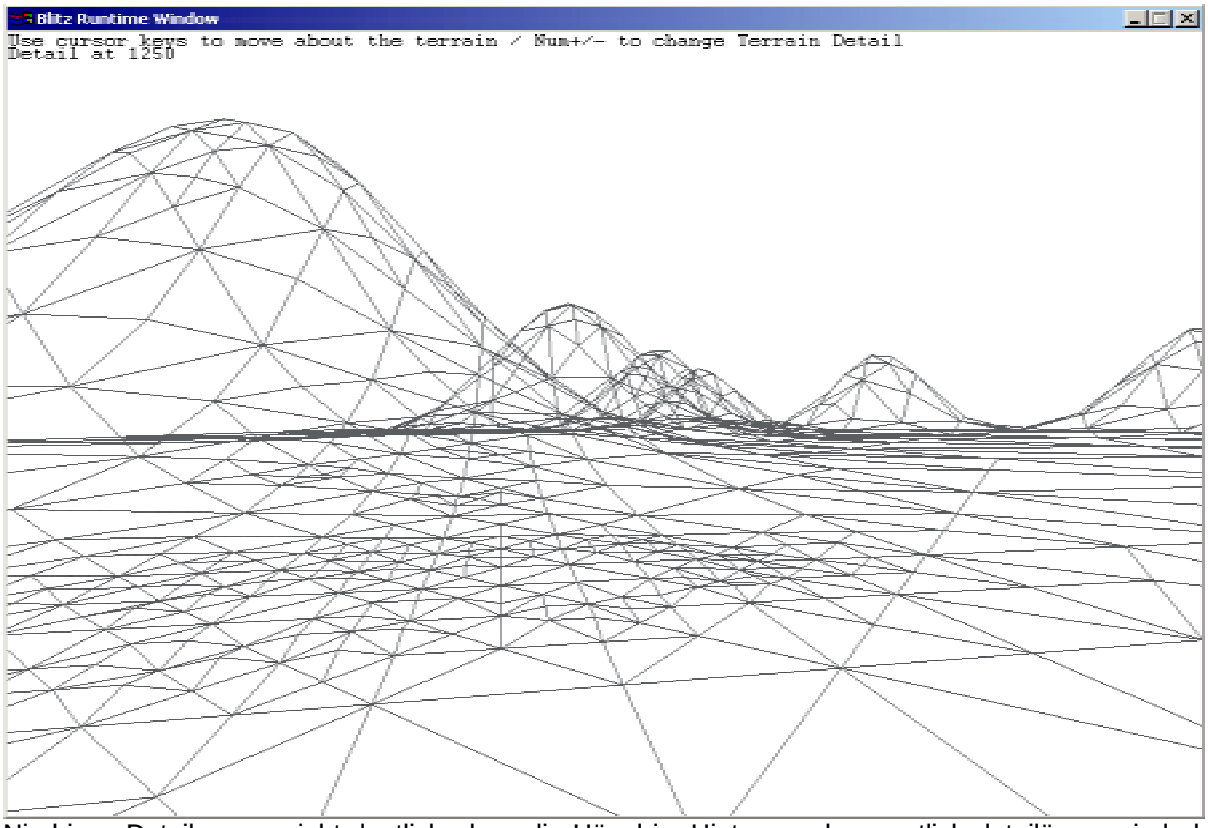
Terrain in Blitz3D

LOD - Level of Detail

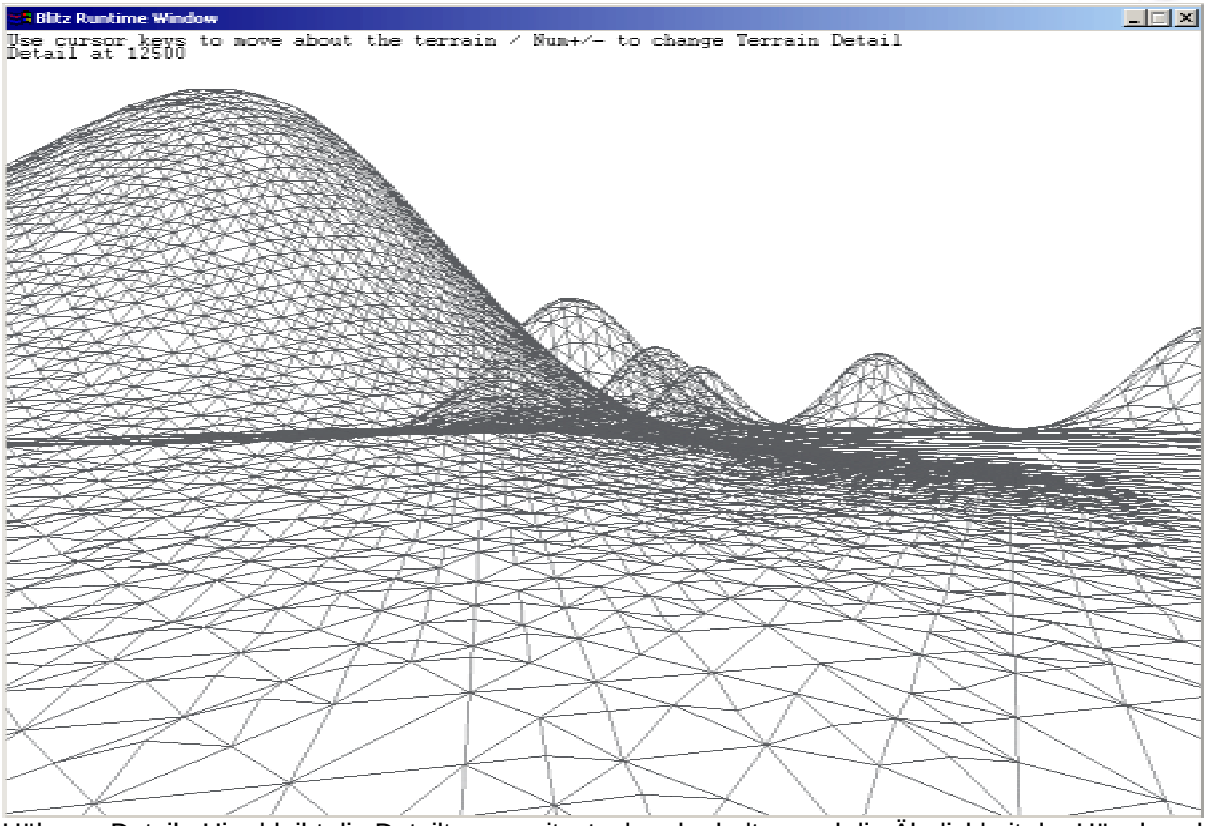
Syntax: **TerrainDetail** *terrainhandle, int Detail, Boolean Vertexmorphing*

Die automatische Regulierung des Level of Detail ist wohl eines der wichtigsten Features, welche die durchaus hohe Performance der 3D-Umgebung garantiert. Hierbei werden entsprechend der Entfernung die Anzahl der sichtbaren Faces und Vertices reguliert. Beeinflussen lässt sich hierbei vom Programmierer nur das Grunddetail in Form eines Integerwertes. So lassen sich zum Beispiel Terraindaten so modifizieren, dass sie auch von älteren Systemen flüssig dargestellt werden können. Am Wireframemodell lassen sich diese Detailunterschiede sehr anschaulich verdeutlichen, wobei man in niedrigen Detailstufen das „Nachmodellieren“ des Terrains sehr gut erkennen kann.

Beispiel für LOD



Niedriges Detail - man sieht deutlich, dass die Hügel im Hintergrund wesentlich detailärmer sind als die im Vordergrund und die Anzahl der Vertices zum Vordergrund stark zunimmt.



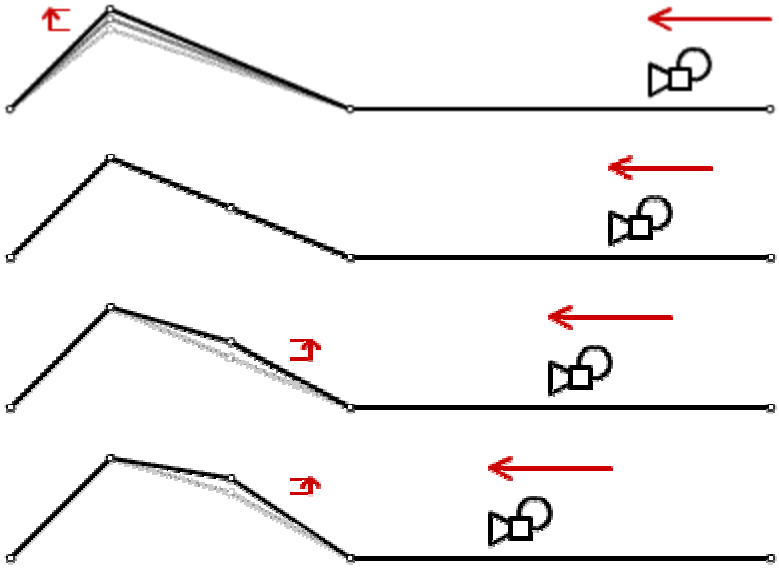
Höheres Detail - Hier bleibt die Detailtreue weitestgehend erhalten und die Ähnlichkeit der Hügel auch in größerer Entfernung ersichtlich.

Vertexmorphing

Ein wesentlicher Bestandteil der Detailfunktion ist das Vertexmorphing. Hierbei wird die Position von Vertexpunkten dynamisch interpoliert, wodurch es nicht zum „flattern“ oder „springen“ kommt, wenn durch Hinzufügen von weiteren Vertexpunkten die Form immer detaillierter wird. Dadurch wird das vereinfachte Modell an das detaillierte in einem weichem Übergang angepasst, was für den Betrachter dann weniger ersichtlich ist. Um das Beschriebene etwas besser zu unterstreichen; hier das Ganze noch einmal vereinfacht dargestellt:
Gegeben sei folgendes Terrain im Querschnitt...

Berücksichtigen wir nun also, dass das Terrain auf Grund der Entfernung zur Kamera angepasst wird. Dadurch haben wir nun auch eine Reduktion an Vertexpunkten im Bereich des Hügels.

Bewegen wir uns nun mit der Kamera dichter an den Hügel heran werden die Vertexpunkte in Anzahl, Höhe und Position immer mehr an das von uns gewollte Modell herangeführt, wie man in den nachfolgenden Abbildungen sehen kann...



Modifizieren von Terrain

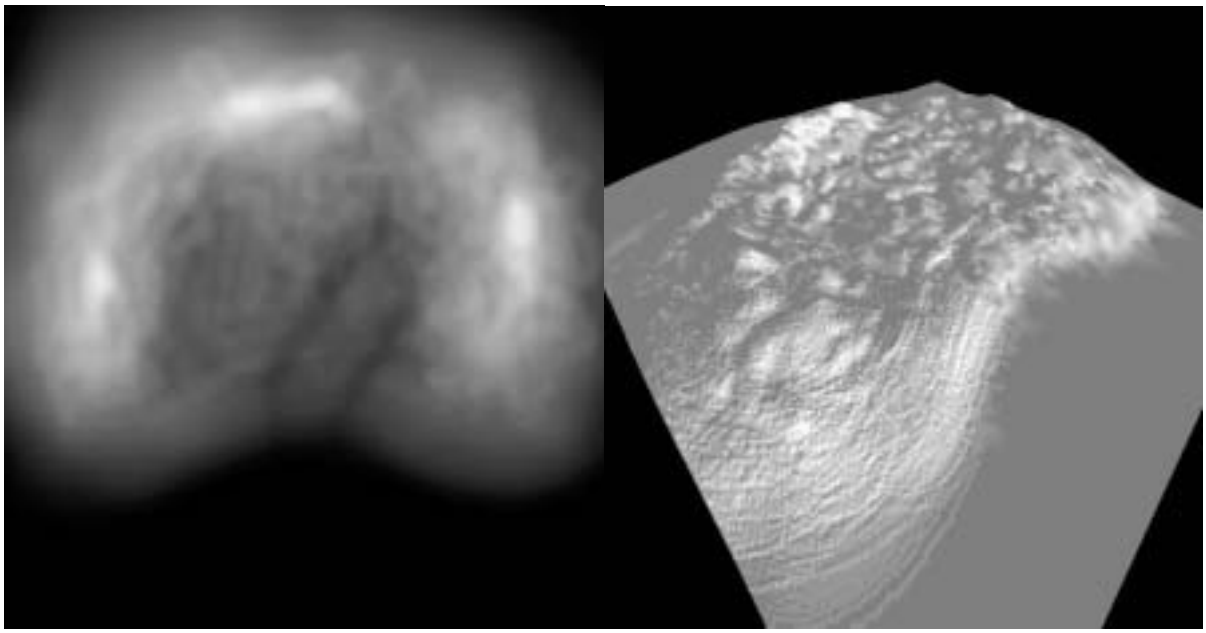
Syntax: **ModifyTerrain** *terrain, int grid_x, int grid_z, float height, [Boolean realtime]*

Das Modifizieren von Terrain gehörte zu einem der ersten faszinierenden Features von Blitz3D, die mir ins Auge stachen, als ich mich mit den Demoprogrammen von der Blitzhomepage beschäftigte. Noch erstaunter war ich, wie einfach und elegant es bei Blitz3D gelöst wurde. Blitz3D bietet hierfür ein kompaktes Paket in Form der Funktion *ModifyTerrain* an. Dieser Befehl macht nichts Anderes, als für einen bestimmten Punkt des Gitters die Höheninformation zu ändern. Das Gitter entspricht in Ausdehnung und Knotenpunkten entweder den in *CreateTerrain* festgelegten Werten oder der Pixelzahl einer als Höhenkarte fungierenden Bitmapvorlage.

Wenn man zum Beispiel mittels *LoadTerrain* eine Höhenkarte lädt, welche eine Größe von 512x512 Pixel hat, besitzt das Terrain dann auch 512x512 Gitterpunkte, unabhängig von dessen späteren Skalierung.

Eine Höhenkarte oder auch Heightmap ist ein Graustufen-Bitmap wobei der Grauwert die Höhenposition wiederspiegelt. Also Weiß = Hoch, Schwarz = Niedrig.

Beispiel für eine Heightmap:



Links sehen wir das Bitmap das als Höhenkarte diente, rechts das Resultat.

Der Echtzeitparameter am Ende der Funktion beeinflusst die Umsetzung dieser Funktion. Normalerweise wird diese Funktion erst mit dem Befehl *Renderworld* umgesetzt, da hierbei alle Objekte in der Welt gerendert werden.
Ist dieser Wert jedoch auf TRUE gesetzt, wird die Veränderung schnellstmöglich, eben in Echtzeit ausgeführt. Dadurch können zum Beispiel ungenaue Kollisionsabfragen verhindert werden.
Die Höhenangabe des *ModifyTerrain*-Befehls ist ein Fliesskommawert zwischen 0 und 1, wobei 0 Flach und 1 Hoch ist. Die tatsächliche Höhe auf der Karte ist später von der Stärke der Skalierung der Y-Achse abhängig.
Hat man also die Höheninformation 100 skaliert, entspricht ein Wert von 0.7 im Höhenparameter des *ModifyTerrain*-Befehls einer späteren tatsächlichen Höhe von 70 von 100 Einheiten[1]. Da der Höhenparameter absolut ist und nicht relativ zur aktuellen Höhe sollte man z.B. bei der Kollision von einem Objekt mit dem Terrain die aktuelle Höhe an der Aufschlagposition ermitteln.
Hierfür bietet Blitz3D zwei Funktionen an, welche die Höhe des Terrains an einer bestimmten Position wiedergeben.
Die eine Funktion ist *float TerrainHeight(terrain, grid_x, grid_z).* Diese Funktion gibt die Höhe des Terrains an Punkt X,Z in einem Fliesskommawert zwischen 0 und 1 zurück. Dadurch ist sie bestens für unser Beispiel geeignet, da wir hier keine Werte zurückrechnen müssen.
Anmerken möchte ich hierbei nur, dass z.B. die Kameraposition in einer skalierten Karte nicht zwangsläufig mit den Gitterkoordinaten des Terrains übereinstimmen muss. Da sich ja mit dem Skalieren nicht die Anzahl der Gitterknoten ändert, wohl aber die Fläche, muss man die Position der Kamera dann mit dem Skalierungsfaktor auf die Ursprungskoordinaten des unskalierten Terrains zurückrechnen.
Möchte man daher z.B. die Kamera auf das Terrain abgleichen, z.B. immer in einer konstanten Höhe über der Karte schweben, ist die zweite Funktion
float TerrainY(terrain, x, y, z) zur Höhenbestimmung des Terrains besser geeignet.

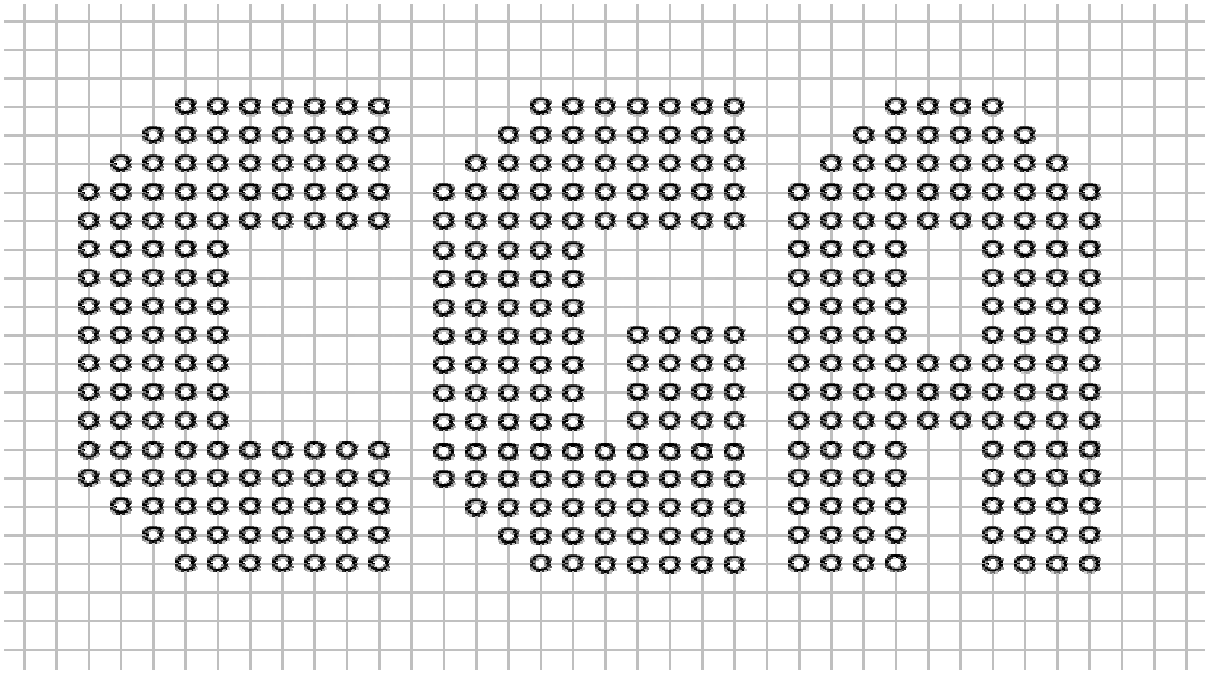
TerrainY liefert hierbei die Höheninformation basierend auf dem aktuellen Stand des Terrains in der Welt. So können wir zum Beispiel mit unserer aktuellen Kameraposition die Höhe des Terrains an dieser Stelle erfragen.

[1] Eine Einheit entspricht der Länge des Einheitsvektors der entsprechenden Achse

Als Rückgabewert liefert diese Funktion einen Fliesskommawert, welcher der aktuellen skalierten Höhe des Terrains an dieser Stelle entspricht. Ist der Wert einer mit dem Faktor 100 skalierten Höhe 38 Einheiten, so liefert uns *TerrainY* auch 38.00000 zurück, TerrainHeight jedoch lediglich 0.38.

Kommen wir zu einem etwas praktischeren Beispiel der Terrainmodifizierung.
(*siehe auch Terrain_DEMO.bb/Terrain_DEMO.exe*)

Zur Veranschaulichung der Modifizierung des Terrain habe ich innerhalb einer Ebene die Buchstaben CGA herausgestanzt. Dafür habe ich mir zu erst die Buchstaben auf ein Gitter aufgezeichnet um die Funktionsweise auf das Gitter eines Terrains besser zeigen zu können, da dies im Prinzip der Draufsicht entspricht.



Auf Grund der Fülle von Gitterknoten war es dann auch sinnvoll für jeden dieser Buchstaben ein zweidimensionales Feld der Größe 17x10 anzulegen.

Dim C(16,9) ;definiert ein zweidimensionales Feld von 0..16 und 0..9
Dim G(16,9)
Dim A(16,9)

Als nächstes sollten diese Felder mit Daten gefüllt werden. In Blitz3D passiert das durch eine simple Zuordnung, wobei Blitz mehrere Zuordnungen für sich getrennt in einer Zeile gerne toleriert.

; C Matrix füllen

C(0,0)=0 C(0,1)=0 C(0,2)=0 C(0,3)=1 C(0,4)=1 C(0,5)=1 C(0,6)=1 C(0,7)=1 C(0,8)=1 C(0,9)=1

Diese Zuordnung erfolgt auch für die anderen 16 Reihen um die jeweilige Matrix zu komplettieren. Fehlt nur noch das eigentliche Modifizieren.
Durch das Verschachteln zweier *For...Next* Schleifen können wir nun die Matrizen der Buchstaben auslesen und an die Funktion *ModifyTerrain* weiterleiten.
Um das Ergebnis besser betrachten zu können habe ich die Buchstaben nur um 30 Einheiten herausgehoben.

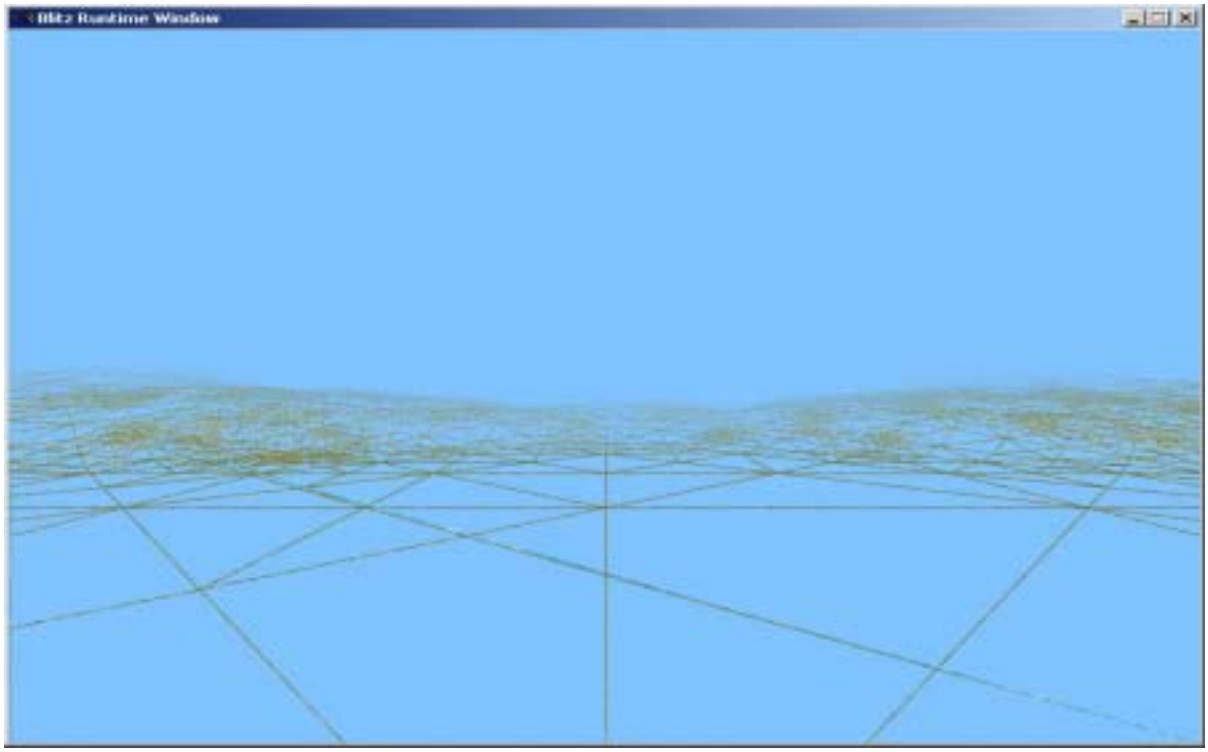
For i= 0 To 16

```
  For j= 0 To 9
    If C(i,j)=1 Then ModifyTerrain terrain,(x-16)+j,(y+8)-i,0.3,1
    If G(i,j)=1 Then ModifyTerrain terrain,(x-5)+j,(y+8)-i,0.3,1
    If A(i,j)=1 Then ModifyTerrain terrain,(x+6)+j,(y+8)-i,0.3,1
  Next
Next
```

Die Werte, die zu den x- und y-Werten, an denen die Buchstaben erscheinen sollen, hinzuaddiert bzw. subtrahiert werden, basieren auf dem Abstand der Buchstaben von einander und vom Anzeigemittelpunkt. Auch wenn bei dieser Anwendung eigentlich keine Echtzeitanzeigen von Nöten sind, habe ich mich bei dieser Demonstration dafür entschieden, da ich einen entscheidenden Performancegewinn bei der Echtzeitanzeige festgestellt habe. Vor allem wenn die Verformung in mehreren Schritten animiert wird und nicht durch eine einzige Wertänderung gleich ihre Endposition erreicht. Dadurch schien das Ergebnis wesentlich flüssiger und nicht so stufig.

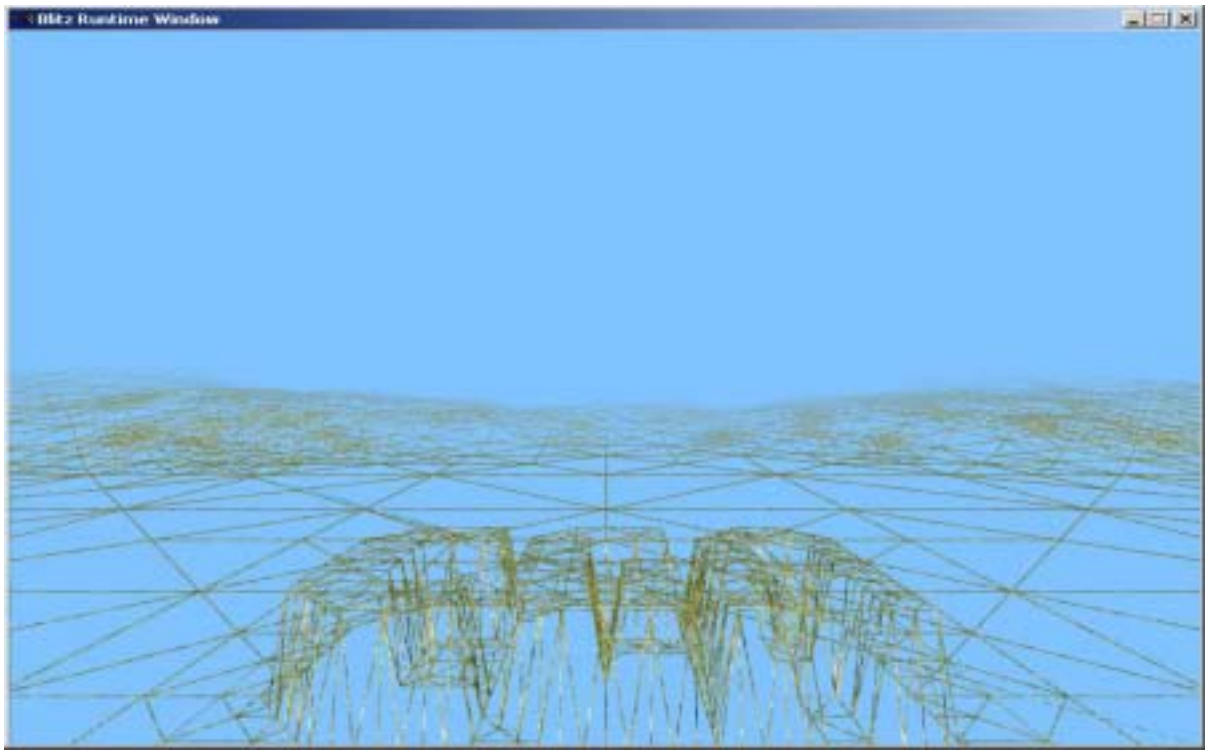
Hier nun ein paar Abbildungen der Resultate aus dem Programm:

Die Szene / Wireframed:



Das Terrain vor der Manipulation, die Grobstruktur des Vordergrundes basiert lediglich auf dem Fakt, dass es sich hier um eine Ebene ohne Höhenunterschied handelt und hier eine größere Anzahl von Faces eine reine Verschwendung von Rechenleistung gewesen wäre.

Die modifizierte Szene / Wireframed:



Dieselbe Szene nach der Manipulation. Man sieht deutlich die Erhöhung der Anzahl von Vertexpunkten im Vordergrund, wo vorher deutlich reduziert wurde.

Die Szene noch einmal ohne Wireframes:

Hier sehen wir noch einmal die selbe Szene mit Texturen und ohne Drahtgittermodell.

Texturen in Blitz3D

Texturarten

Im diesem Themengebiet möchte ich gerne die Texturarten in Blitz3D vorstellen.
In Blitz3D gibt es drei grundlegende Arten von Texturen.

1. Standardtexturen
2. animierte Texturen
3. dynamische (laufzeitgenerierte) Texturen

Standardtexturen

Als Standardtexturen möchte ich hier statische Bilder bezeichnen. Man kann sie mittels *„LoadTexture (Dateiname$[2], flag)“* laden und Blitz3D unterstützt eine große Menge der häufig genutzten Bildformate, wie z.B. .bmp, .jpg, .gif, .png, .tga etc.

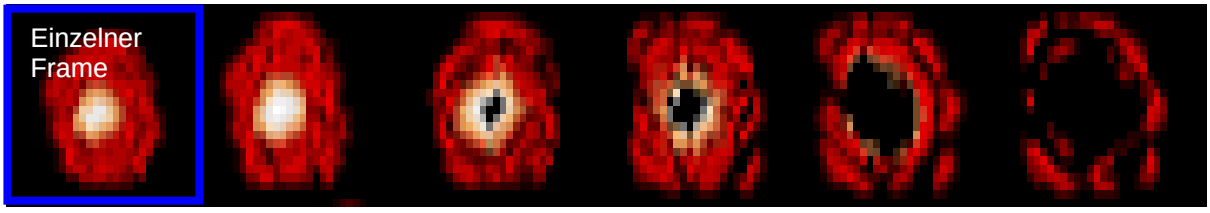
Die Flags dieser Funktion sind optional und geben das Texturverhalten von Blitz3D vor. So kann zum Beispiel eine Textur als Mipmapped-Textur gelesen werden, um auf größerer Distanz heruntergerechnete Instanzen dieser Textur zu nutzen, um Rechenleistung und Speicher zu sparen. Texturen können aber auch als Reflectionmap geladen werden was bedeutet, das diese sich nicht der Ausrichtung des Objektes anpassen, sondern auf diese Weise den Eindruck vermittelt das sich die starre Umgebung in dem Objekt spiegelt. Hier die Texturflags noch einmal zur Verdeutlichung:

```
flags (optional) - texture flag:
1:    Color
2:    Alpha
4:    Masked
8:    Mipmapped
16:   Clamp U
32:   Clamp V
64:   Spherical reflection map
```

Kombinationen der Filter sind durchaus möglich. Zum Beispiel 9 (1+8) hieße die Textur wäre mipmapped und Color Map.
Diese Eigenschaften lassen sich jedoch auch nachträglich mittels Texturfilter einstellen.

Animierte Texturen

Animierte Texturen sind wirklich ein starkes Feature einer jeden 3D-Umgebung. Sie vermögen es, einer Szenerie Leben einzuhauchen und schaffen erst das richtige Ambiente. Das alles geschieht auf Kosten von relativ wenig Rechenleistung im Vergleich zu bewegten 3D-Strukturen.
Bei Blitz3D kann jedes Bildformat Vorlage für eine animierte Textur sein. Wichtig hierbei ist lediglich, dass die Frames nahtlos aneinandergereiht sind und stets die selbe Größe aufweisen. Ein gutes Beispiel einer animierten Textur ist dieses Bild:



[2] $ bezeichnet eine Variable/Parameter vom Typ String

Hier sieht man deutlich die Reihenfolge der Animation und die gleiche Framegröße.
Geladen werden animierte Texturen mit der Funktion:

LoadAnimTexture(Dateiname$,flags,frame_width,frame_height,first_frame,frame_count).

Die Flags verhalten sich hier genauso wie bei den Standardtexturen, lediglich die Informationen über Höhe, Weite und Anzahl der Frames kommen hier neu hinzu.

Nehmen wir obenstehende Animation als Beispiel um die Verwendung dieser Funktion kurz zu verdeutlichen:

LoadAnimTexture(„explode.bmp“,8,20,20,0,6).

20x20 ist hier die Framegröße, 0 bedeutet dass die Animation mit dem allerersten Frame beginnt, die 6 ist die Anzahl der Frames.
Um eine Animation später abzuspielen, kann man bei der Zuweisung mittels

EntityTexture entity$,textur$[,frame,index]

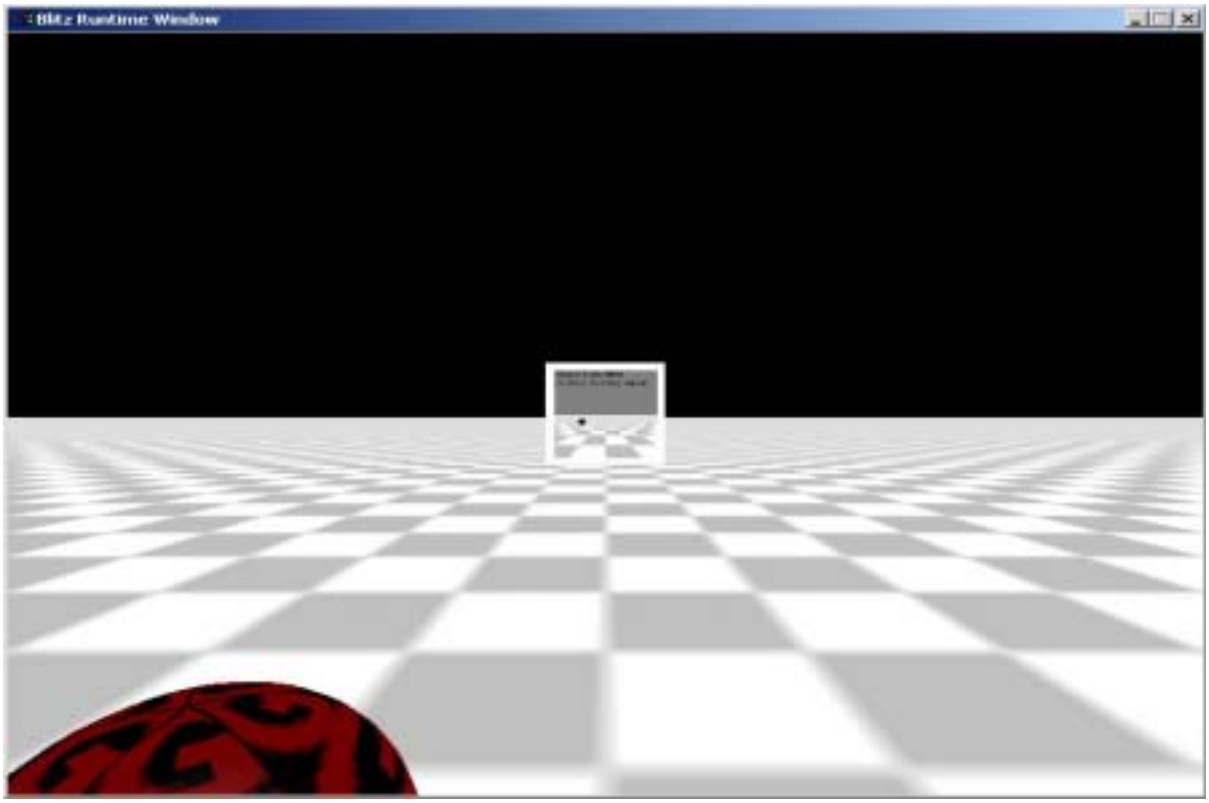
den entsprechenden Frame der Animation aufrufen. Wobei der Parameter *frame* durch die Nummer des anzuzeigenden Frames ersetzt wird; beginnend mit 0.
Ein Beispiel hierzu findet sich auch in der „Dynamic Textures Demo“.

Dynamische Texturen

Dynamische Texturen sind zwar nicht neu, aber trotzdem ein Feature mit dem nicht jede 3D-Engine heutzutage aufwartet. Man trifft auf dieses Feature zum Beispiel in Fahrzeug/Rennsimulationen wo häufig Spiegelungen auftreten, sei es auf dem Lack oder in einem Rückspiegel. Überall wo man realistische Spiegelungen möchte und statische Reflectionmaps nicht ausreichen, kommt man nicht darum herum, Texturen zur Laufzeit zu erzeugen, welche die aktuelle Umgebung wiederspiegeln.

FPS (First Person Shooter) warten diesbezüglich oft mit Tricks auf um Spiegelflächen zu simulieren. Hier wird dabei ein identischer Raum spiegelverkehrt hinter einer „Scheibe“ modelliert und man projiziert die 3D-Objekte des aktuellen Raumes ebenfalls in den „Spiegel“-Raum. Diese Technik wurde zum Beispiel bei „Duke Nukem 3D“ verwandt.
Bei Blitz3D ist es allerdings möglich, einen leeren Texturbuffer im Speicher zu erstellen und eine Kamera in diesem Speicherbereich seine Bilddaten ablegen zu lassen, wodurch die Textur immer die entsprechende Kamerasicht wiederspiegelt bzw. darstellt.

Betrachten wir folgende Abbildung dazu:



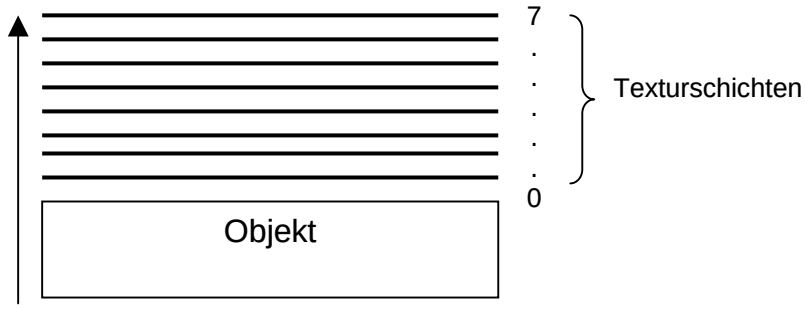
Hier sehen wir einen weißen Quader mit einer Spiegelfläche welche die Kugel im Vordergrund exakt wiederspiegelt.
Die Technik hierbei ist eigentlich relativ simpel. Eine zweite Kamera, welche hier entgegengesetzt der Betrachterkamera ausgerichtet ist, „beobachtet“ die Geschehnisse aus der Sicht des Spiegels und schreibt die Bildinformationen in den Speicherbereich der Textur, die sich auf einem kleineren Quader innerhalb des großen Quaders befindet.
Damit das Bild auch seitenverkehrt ist und nicht so wie es die andere Kamera wirklich wahrnimmt, skalieren wir die Textur an der X-Achse um einen negativen Wert. Die Möglichkeit der Erstellung von Texturen zur Laufzeit bringt zum Beispiel in der Onlinespielindustrie entscheidende Vorteile. Können doch so durch 2D-Tools wie Linien, Flächen etc. Zufallstexturen erstellt werden die nicht den Nutzer zwingen, erst dutzende MegaBytes aus dem Internet zu laden. Denn schließlich werden oft mindestens 40% des Speicherbedarfs von Spielen nur durch Texturen verwendet. Die Schachbretttextur auf dem Boden wurde auch zur Laufzeit erstellt und hat keine Bitmapvorlage.

Multitexturing

Ein essentieller Bestandteil moderner 3D-Engines ist das Multitexturing. Durch dieses Verfahren kann man Objekten zum Beispiel Glanz verleihen, Texturen mischen um mehr Struktur zu erzeugen, Reflektionen simulieren oder kleine Details addieren, die eine Textur realistischer und einzigartiger wirken lassen.
Blitz3D bietet dabei für jedes Objekt acht Schichten an, in denen Texturen abgelegt werden können. Zugewiesen bekommt eine Textur dabei die Schicht bei der Zuordnung zum Objekt. Und zwar mit der *EntityTextur*-Funktion die ja syntaktisch schon in dem Abschnitt über die Animierte Textur erwähnt wurde.

EntityTexture entity$,textur$[,frame,index]

Gehen wir nun etwas mehr auf den letzten Parameter ein. Der Index weist der Textur eine eindeutige Position auf dem Objekt zu.



So wird also der untersten Schicht (0) eine Textur durch:

EntityTexture object,basetex,0,0 zugewiesen.

Da der Parameter *Frame* vor dem Index kommt müssen wir ihn, wenn es sich um keine animierte Textur handelt, auf 0 setzen, da ja jede Textur mindestens einen Frame besitzt.

EntityTexture object,toptex,0,7 bildet also die Spitze des Texturstapels. Texturen werden standardmäßig immer multipliziert. Damit am Ende von sieben Texturen kein schwarzes Objekt herauskommt, muss man also die Eigenschaften der Textur ändern. Hierbei bietet Blitz3D die Funktion

TextureBlend textur$,blendmode an.

Diese Funktion ermöglicht es uns zu entscheiden ob eine Textur addiert, multipliziert, deren Alphakanal genutzt oder sie gar nicht gezeigt werden soll.

0: disable texture
1: alpha
2: multiply (default)
3: add

Standard ist also *TextureBlend textur,2.* Beachten sollte man hierbei die Reihenfolge der Texturen später auf dem Objekt. Schließlich bringt es nichts, die unterste Textur zu addieren. (Beispiele hierfür finden sich in der Terrain_DEMO.exe / Terrain_DEMO.bb.)

Speicherverwaltung

Als letzten Punkt der Texturen in Blitz3D möchte ich noch bemerken, dass alle Texturen sich zu jedem Zeitpunkt neu zuweisen, neu laden und, was die Performance vor allem im Speicherbereich betrifft, sich auch aus dem Speicher entfernen lassen. Wenn eine Textur geladen wird bedeutet das, dass sie zu jeder Zeit des Programms verfügbar ist, egal für welches Objekt. Das wiederum hat zur Folge dass nicht nur der durch das texturierte Objekt benötigte Speicher belegt wird, sondern auch zusätzlicher Speicher in dem die Textur bereit gehalten wird verbraucht wird. Dadurch bietet Blitz3D das Feature diese Texturdaten freizugeben, wodurch wertvoller Speicher wieder verfügbar wird. Das texturierte Objekt wird dabei nicht davon betroffen. Jedoch lässt sich die Textur auf dem Objekt dann auch nicht mehr beeinflussen, z.B. in Ausrichtung, Skalierung, Filterung etc., weil das Texturobjekt als solches ja dann nicht mehr existiert. Was allerdings nicht heißen soll, dass ein Objekt dessen Textur freigegeben wurde nicht mehr anderweitig texturiert werden kann.

So geht es übrigens mit vielen Objekten bei Blitz3D, zum Beispiel Schriftarten die zu einem bestimmten Zeitpunkt nicht mehr gebraucht werden, lassen sich ebenso entladen wie Texturen.

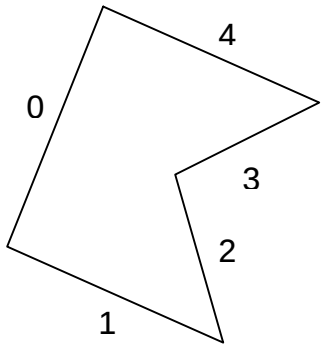
BSP Support unter Blitz3D

Was ist BSP – Eine kleine Einführung in die BSP-Bäume

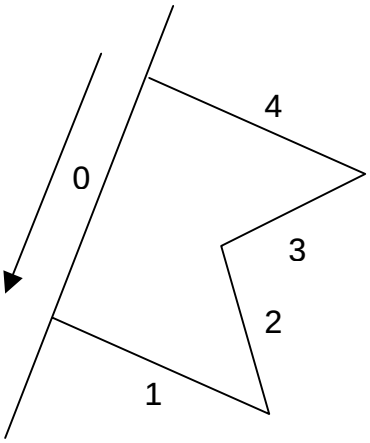
BSP steht für Binary Space Partitioning. Mit anderen Worten: Teile einen Raum in zwei Teile und nutze dabei eine Baumstruktur. BSP-Bäume werden in diversen Anwendungsbereichen genutzt. So zum Beispiel bei der Kollisionsabfrage, der Tiefensortierung, bei der Bestimmung von der Sichtbarkeit von Oberflächen, dem Beschreiben von Polygonflächen und vielen anderen. Es gibt zwei Arten von BSP-Bäumen, Node-based trees (Knoten-basierte Bäume) und Leaf-based trees (Blatt-basierte Bäume).

Um die Raumteilung und Baumstruktur besser erläutern zu können, werde ich mich der Beschreibung eines Polygons bedienen.

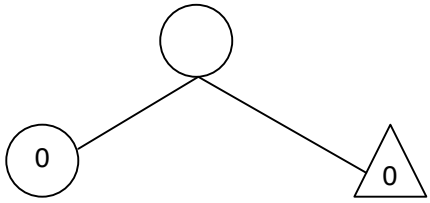
Fangen wir also mit dem Polygon an. Gegeben sei folgendes fünfseitige Polygon:



Um nun die eingeschlossene Fläche zu beschreiben, betrachten wir uns einmal die einzelnen Kanten.

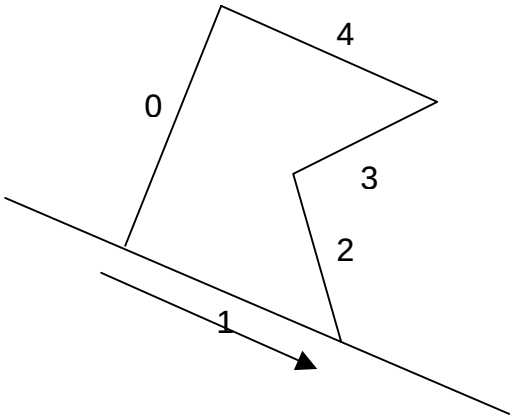


Beginnen wir bei unserer Betrachtung mit Kante 0. Verlängert man diese Kante, erkennt man sehr schnell, das sie den Raum in zwei Seiten teilt. Einmal die Fläche links von dieser Kante und einmal die Fläche rechts davon.
Auch stellen wir fest, das in der rechten Fläche keine Objekte liegen, in der linken aber sehr wohl.
Also lasst uns das grafisch festhalten.
Wir haben einen Ausgangspunkt, das Polygon ...



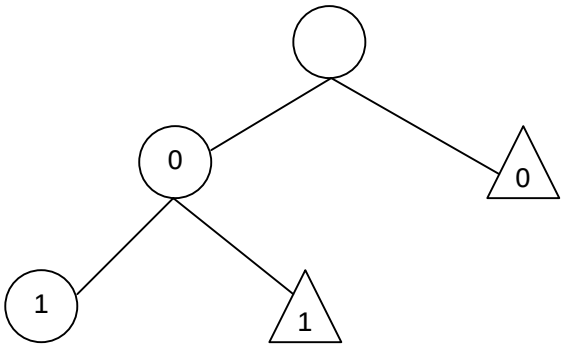
Und 2 Seiten von Kante 0 aus...

Zur Symbolik. In diesem Baumdiagramm bedeutet ein Kreis -> auf dieser Seite liegt ein Objekt, und ein Dreieck bedeutet -> außen.

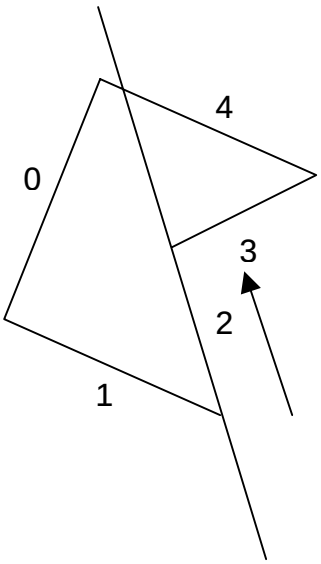


Betrachten wir die nächste Kante: Kante 1. Auch sie teilt verlängert den Raum in zwei Teile. Und auch hier enthält, wie bei Kante 0, die rechte Seite keine weiteren Objekte, jedoch die linke Seite sehr wohl.
Da sich Kante 1 auf der linken Seite von Kante 0 befindet, tragen wir sie dort ein. Also vervollständigen wir unser Diagramm.

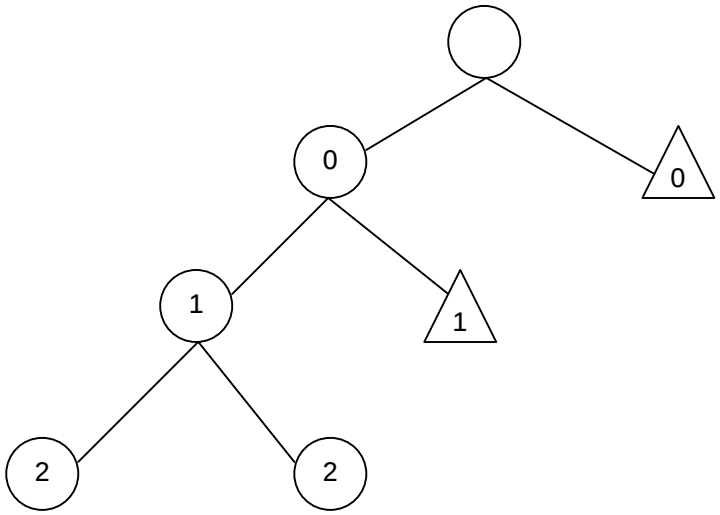
Der Pfeil entlang der Polygonkante dient hier als kleine Hilfe, und stellt die Betrachtungsrichtung dar.



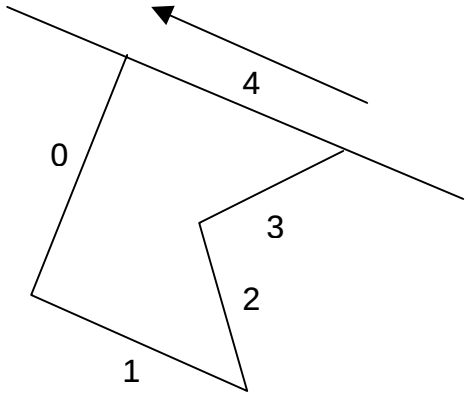
Gut, wie wir sehen, wächst langsam ein Baum aus unserem Diagramm. Betrachten wir Kante 2.



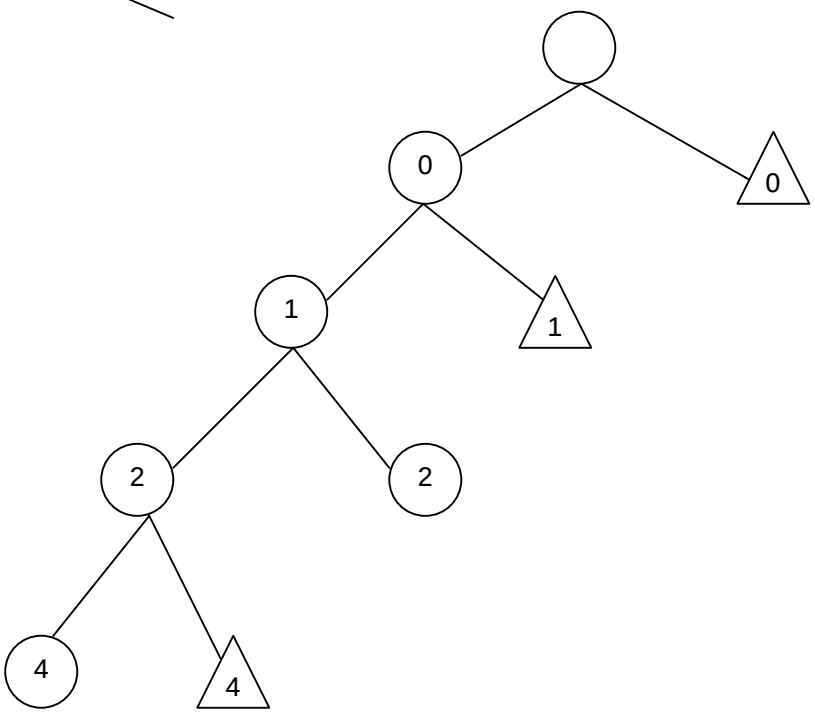
Hier sehen wir nun, dass sowohl auf der rechten als auch auf der linken Seite Objekte vorhanden sind. Bereits betrachtete Objekte/Kanten, werden aber hierbei nicht berücksichtigt. So scheint es auf den ersten Blick auf der linken Seite keine neuen Objekte zu geben. Jedoch schneidet unsere Trennlinie die Kante 4. Das heißt sie ist auf beiden Seiten vorhanden. Demnach besitzen beide Seiten weitere neue Objekte. Halten wir das fest.



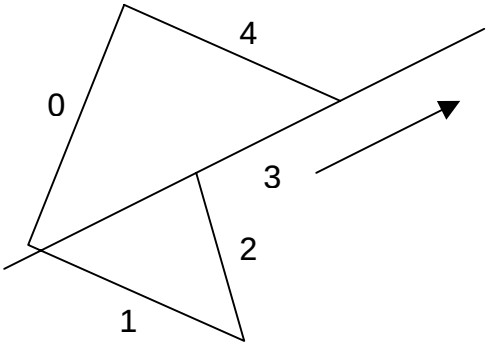
Sowohl die linke als auch die rechte Seite enthielten weitere Objekte, die noch nicht betrachtet wurden. Daher bekommen auch beide einen Kreis. Verfolgen wir also zuerst den linken Ast weiter. Auf der linken Seite ist das nächste noch nicht betrachtete Objekt, Kante 4. Also betrachten wir uns diese genauer.



Die Kante 4 als letzte Kante hat auf der linken Seite nur bereits betrachtete Objekte. Ausser Kante 3, da wir aber nur Kanten aufsteigend betrachten ist Kante 3 für uns nicht weiter von Relevanz. Also finden wir hier einen Abschluss dieses Astes. Und tragen ihn auch als solchen ein.

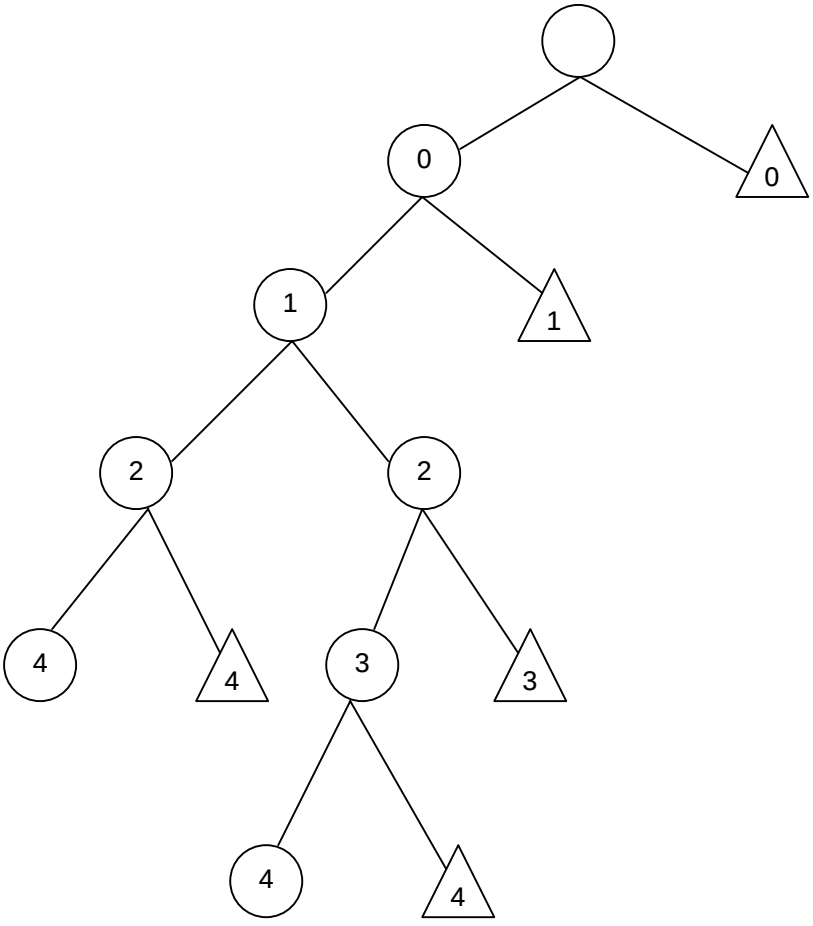


Gehen wir nun zurück und betrachten den rechten Ast von Kante 2. Bei ihr ist das nächste Objekt Kante 3. Also betrachten wir nun Kante 3.



Teilen wir den Raum erneut entlang der Kante 3. Auf der rechten Seite von Kante 3 gibt es keine neuen Objekte und es ist die Außenseite des Polygons. Die linke Seite enthält aber ein neues Objekt, Kante 4. Diese wurde, wenn wir von hier aus den Ast bis zur Wurzel verfolgen hier noch nicht betrachtet. Also haben wir hier einen Anschluss und aktualisieren unser Baumdiagramm. Da wir Kante 4 aber bereits vorher betrachtet hatten wissen wir, dass es bei Kante 4 keine neuen Objekte gibt also schließen wir unser Diagramm gleich ab.

Dadurch kommen wir auf die folgende Baumstruktur.



Betrachten wir nun den Baum von seiner Wurzel bis zu den Blättern, können wir das Polygon nun beschreiben. Zwar nicht in seiner exakten Größe, aber in seiner Form.

In Spielen, dessen Karten auf dieser BSP-Baumstruktur basieren, werden weniger die Polygonflächen durch BSP-Bäume beschrieben. Hier bilden konvexe Regionen die Blätter eines BSP-Baumes. Wobei jedes Mal, wenn die Kamera in einem Level platziert wird, sie

sich in genau einer dieser konvexen Regionen befindet. Diese Blätter wiederum sind gruppiert mit benachbarten Blättern und bilden dadurch Cluster. Wie die Cluster jedoch speziell gebildet werden ist immer abhängig von dem Tool das die BSP-Dateien erstellt. Für jedes dieser Cluster gibt es eine Liste, in der alle weiteren Cluster die möglicherweise sichtbar sind, gespeichert sind. Das wird auch als PVS (potentially visible set) bezeichnet.

Um nun eine solche Karte zu rendern, wird zuerst der BSP-Baum traversiert um festzustellen, in welchem Blatt sich die Kamera befindet. Nachdem wir das herausgefunden haben, wissen wir auch in welchem Cluster sie sich befindet, da ja

jedes Blatt nur in genau EINEM Cluster enthalten ist. Dann wird das PVS für diesen Cluster entpackt, das uns eine Liste aller möglicherweise sichtbaren Cluster von der Kameraposition aus liefert. Jedes Blatt beinhaltet eine Begrenzungsbox, welche genutzt wird um schnell Blätter zu entfernen, die nicht innerhalb des Sichtfeldes sind.

BSP unter Blitz3D

Da wir uns nun dem Ende dieser Semesterarbeit nähern möchte ich abschließend noch etwas auf die Unterstützung von BSP in Blitz3D eingehen.

Mittlerweile gibt es Dutzende Tools von Drittanbietern zur Levelgenerierung von BSP Dateien. Das liegt mit daran, dass die Q3-Engine eine der bekanntesten 3D-Engines ist, die momentan auf dem Markt kursiert. Viele der heute aktuellen Shooter basieren auf der Q3-Engine, seien es “Jedi Knight 2”, “Medal of Honor” oder “Soldier of Fortune 2”. Blitz Research Ltd. hat dieses Konzept aufgegriffen, um so flexibel wie möglich zu sein. Da Blitzbasic oder auch Blitz3D von Haus aus keine Tools mitbringen, um Levelmaps zu generieren, bleiben hierfür eigentlich nur drei Alternativen. Ein 3D-Studio-Modell, ein DirectX-Modell oder ein BSP-Modell. Da jedoch nicht jeder über 3D-Modelling-Software verfügt, jedoch Leveleditoren für Q3-basierte Spiele oft sogar kostenlos verfügbar sind, ist der Vorteil dieser Variante nicht von der Hand zu weisen.

Nachteilig ist im Moment nur, dass die Möglichkeiten BSP-Modelle zu handhaben noch äußerst begrenzt sind und nur im Internet ausreichend dokumentiert werden. Die Kontexthilfe in der IDE ist diesbezüglich leider noch nicht auf dem aktuellsten Stand. Auch sind die Resultate des BSP-Modells sehr von dem Tool abhängig, mit dem man dieses erstellt. Blitz3D beschränkt derzeit den Support von BSP-Dateien nur auf die Q3-Generation, wobei Blitz3D mit Karten aktuellerer Spiele leichte Probleme hatte. Das ist allerdings auch auf den Extraktor zurückzuführen, welcher ebenfalls trotz angegebener Unterstützung einiger Titel Probleme hatte, den integrierten Leveleditor sauber mit diesen Karten zu starten.
Jedoch funktioniert die Karte, welche in der BSP Demo geladen wird, ausgezeichnet.
Geladen wird ein BSP-Level durch folgende Funktion:

LoadBSP(Dateiname$[, float Gamma][, parent])

Der Gammawert ist ein Fliesskommawert zwischen 0 (Standard) und 1 (volle Helligkeit) jedoch wirkt sich dieser nur auf die Lightmaps aus. Wenn das BSP-Modell nur mit Vertexlighting betrieben wird, hat der hier eingestellte Gammawert keinen Einfluss. Falls ein Parentobjekt angegeben wurde, wird das BSP-Modell an dessen Position erstellt.

Zwei weitere Funktionen, die Blitz3D für die Manipulation eines BSP-Modells anbietet sind,

BSPAmbientLight bsphandle, int rot, int gruen, int blau und

BSPLighting bsphandle, boolean lightmap

BSPAmbientLight setzt das Umgebungslicht des BSP-Modells, sowohl für Vertexlighting als auch für Lightmaps. Wobei *BSPLighting* klärt, ob für das BSP-Modell Lightmaps genutzt werden sollen oder nicht. Hierbei ist VertexLighting der Defaultwert, sei es denn aus Performancegründen oder der Einfachheit halber.

Des weiteren können BSP-Objekte wie andere Objekte auch skaliert, verschoben oder rotiert werden. Allerdings können BSP-Objekte nicht nachträglich in Blitz texturiert werden. Texturen werden jedoch automatisch geladen, wenn sie den Pfadangaben entsprechend beiliegen. Alles in allem ist der gegebene Support von BSP-Dateien schon recht beeindruckend, jedoch fehlt es meiner Meinung nach noch ein wenig an den Feinheiten und Details, um sagen zu können, das Blitz3D das Optimum aus den durch BSP gegebenen Möglichkeiten herauszuholen vermag.

Fazit:

Blitz3D ist eine wirklich beeindruckende Programmiersprache welche es sich zur Aufgabe gemacht hat, durch Performance und KnowHow im 3D Bereich aufzutrumpfen. Es war selten so aufregend eine Programmiersprache auszuprobieren und zu erforschen, da man hier mit wenig Aufwand schnell Resultate sieht und sich auch schnell heimisch fühlt. Zwar mag man dabei skeptisch sein, ob eine Programmiersprache die so einfach erscheint nicht schnell an ihre Grenzen stößt, wenn man erst einmal anfängt tiefer einzusteigen, aber ich denke diese Sprache ist flexibel genug; um von der Kreativität des Programmierers zu profitieren.
Ich möchte hierbei nur einmal auf die neue Demo-Sektion auf der Homepage verweisen wo mittlerweile ziemlich beeindruckende Projekte einen Platz gefunden haben. Sicher ist Blitzbasic nicht so universell wie alt eingesessene Programmiersprachen, wie C++ oder Java. Aber dafür wurde Blitzbasic ja auch nicht konzipiert. Ich kann nur jedem Empfehlen den diese Arbeit oder die Beispielprogramme neugierig gemacht haben, sich die Demo herunterzuladen und sich selbst ein Bild zu machen.

Daniel Backmann, s681573

Quelltexte:

2D_Demo.bb

```
Graphics 800,600,16,1
SetBuffer BackBuffer()                          ;BackBuffer fuer Doublebuffering erzeugen
SetBuffer FrontBuffer()                         ;FrontBuffer aktivieren
Flip                                            ;zwischen den Buffern umschalten

DoubleBuffering()                               ;Doublebuffering Demo

ClsColor 0,0,128                                ;Bildschirmfarbe nach dem Loeschen
Cls                                             ;Loesche Buffer

AnimImage()                                     ;Demonstration animierter Bilder


End


;DoubleBuffering Demo
Function DoubleBuffering()

dbuff=0                                         ;Doublebuffering AUS
x=-20                                           ;X Startwert Sprite
y=(GraphicsHeight()/2)-10                       ;Y Startwert Sprite

While Not KeyHit(1)                             ;Solange ESC nicht gedrueckt wurde tu ...
 Cls                                            ;Loesche aktuellen Buffer
 Text GraphicsWidth()/2,50,"X="+x
 Text 0,0,"DOUBLE BUFFERING DEMO ... PRESS SPACE TO TOGGLE MODE / ESC TO SKIP"
 If dbuff Then
                Text 0,20,"DOUBLE BUFFERING ENABLED"
        Else
                Text 0,20,"DOUBLE BUFFERING DISABLED"
        EndIf

 Rect x,y,20,20,1                               ;Zeichne 20x20 Box an Position x,y

 x=x+1                                          ;Erhoehe X Position um 1
 If x>GraphicsWidth() x=-20                     ;Wenn Bildschirmgrenze ueberschritten setze
                                                ;Boxposition zurueck
 Delay 5                                        ;5 Millisekunden verzoegerung

 If KeyHit(57) dbuff=1-dbuff                    ;LEERTASTE -> 2xBuffermodus AN/AUS

 If dbuff Then Flip                             ;Wenn DBuffermodus AN / schalte Buffer um
Wend
End Function

;Demonstration animierter Bilder
Function AnimImage()
  image=LoadAnimImage("powerup2.bmp",20,20,0,18)    ;Lade animiertes Bild 20x20, 18 Frames
  all=LoadImage("powerup2.bmp")                     ;Lade Bild
  frame=0                                           ;Frameindex auf 0
  x=-20                                             ;X Startwert Sprite
  y=300                                             ;Y Startwert Sprite

While Not KeyHit(1)                                 ;Solange ESC nicht gedrueckt wurde tu ...
  Cls
        ;Loesche Buffer
  Text 0,0,"FRAME="+frame+" ... PRESS ESC TO SKIP"
  DrawImage image,x,y,frame                         ;Zeichne aktuellen Frame von Bild "image" an Position x,y
  DrawImage all,0,50                                ;Zeichne Bild "all" an Position 0,50
  frame=frame+1                                     ;Erhoehe Frameindex um 1
  If frame=18 frame=0                               ;An Frameposition 18 setze Frameindex wieder auf 0

  x=x+3                                             ;Erhoehe X Position um 3
  If x>GraphicsWidth() x=-20                        ;Wenn Bildschirmgrenze ueberschritten setze
                                                    ;Bildposition zurueck

  Delay 50                                          ;Verzoegerung von 50 Millisekunden
```

```
 Flip                                                    ;Buffer umschalten
Wend
End Function

=========================================(ENDE 2D_Demo.bb)=====================================
```

Terrain_DEMO.bb

```
Graphics3D 800,600,16,1
SetBuffer BackBuffer()

Dither True                                             ;Aktiviert Rasterübergänge von Farbverläufen bei 16 Bit Farbtiefe

Global terrain, Animtex, camera, detail, vmorph, water  ;Definieren globaler Variablen

Dim c(16,9)                                             ;Definieren einer 2dimensionalen Liste 17x10
Dim g(16,9)
Dim a(16,9)


; TerrainGroesse
Const ScaleTXZ=2                                        ;Skalierung des Terrains in X und Z Richtung *2
Const ScaleTY=100                                       ;Skalierung des Terrains in Y Richtung *100

; Variablen Vordefinition
Height=GraphicsHeight()
Width=GraphicsWidth()
detail=8000                                             ;TerrainDetail 8000
vmorph=1                                                ;VertexMorphing EIN
help=1                                                  ;Hilfe / Debug Fenster EIN
shade=1
        ;TerrainShading EIN
effect#=0                                               ;Hoeheneffekt 0

Font = LoadFont("Arial",14,1,0,0)                       ;Lade ARIAL Font
SetFont Font                                            ;Setz die aktuelle Schrift auf ARIAL

AntiAlias True                                          ;Antialias EIN

light=CreateLight()                                     ;Erstelle allgemeine Lichtquelle
;PositionEntity light,250,150,250                       ;bei allgemeinem Licht irrelevant sonst verschiebe Lichtquelle
;LightRange light,50                                    ;auch nur interessant z.B. bei Punktlicht, Lichtradius

InitializeTerrain()                                     ; Initialisiere das Gelaende

InitializeCamera()                                      ; Initialisiere die Kamera

SetWater()                                              ; Erstelle Wasser

Wire=0                                                  ; Wireframe AUS

While Not KeyHit(1)                                     ; Solange nicht ESC gedrueckt wurde ... tu...

 If KeyHit(17) Then Wire=1-Wire                         ; W -> Wireframe AN/AUS

 WireFrame Wire                                         ; Setzt den Wireframe Modus

 ; Kamera Höhenjustierung
 If KeyDown(72) Then MoveEntity camera,0,2,0            ; Num8 -> bewege Kamera 2 Einheiten nach oben
 If KeyDown(80) Then MoveEntity camera,0,-2,0           ; Num2 -> bewege Kamera 2 Einheiten nach unten

 ; Behalte die Kamera mind. 10 Einheiten ueber dem Boden
 Y#=TerrainHeight(Terrain,EntityX(Camera)/ScaleTXZ,EntityZ(Camera)/ScaleTXZ)*ScaleTY
 If Y#+10>EntityY(Camera) Then PositionEntity camera,EntityX(camera),Y#+10,EntityZ(camera)
 ; TerrainHeight liefert die aktuelle Hoehe des Terrains an der Position X,Y zurueck

 ; Kamera Rotation
 If KeyDown(75) Then TurnEntity camera,0,2,0
        ; Num4 -> drehe Kamera um 2 Grad um die Y-Achse
 If KeyDown(77) Then TurnEntity camera,0,-2,0
        ; Num6 -> drehe Kamera um -2 Grad um die Y-Achse

 ; Kamera Bewegungen
 If KeyDown(76) Then MoveEntity camera,0,0,2
        ; Num5 -> bewege Kamera 2 Einheiten nach vorn
 If KeyDown(79) Then MoveEntity camera,-2,0,0
        ; Num1 -> bewege Kamera 2 Einheiten nach links
```

```
If KeyDown(81) Then MoveEntity camera,2,0,0
        ; Num3 -> bewege Kamera 2 Einheiten nach rechts
If KeyDown(82) Then MoveEntity camera,0,0,-2
        ; Num0 -> bewege Kamera 2 Einheiten nach hinten

; Terrain modifizieren
If KeyHit(14) Then                              ; Backspace -> Aktiviere Terrainmodifikation
                If Effect=0 Then mode=1
                If Effect=.16 Then mode=2
End If

If (mode=1) Then
                Modify(256,20,Effect)
                Effect#=Effect#+.01
                If Effect#=0.16 mode=0
End If

If (mode=2) Then
                Effect#=Effect#-.01
                If Effect#=0 mode=0
                Modify(256,20,Effect)
End If

; Detail regulieren
If KeyHit(74) Then                              ; Num- -> Detail senken
                detail=detail-500
                If detail<0 detail=0
End If

If KeyHit(78) Then                              ; Num+ -> Detail erhoehen
                detail=detail+500
                If detail>20000 detail=20000
End If

If KeyHit(47) Then vmorph=1-vmorph              ; V -> Vertexmorphing AN/AUS
If KeyHit(20) Then shade=1-shade                ; T -> TerrainShading AN/AUS

TerrainDetail terrain,detail,vmorph             ; Setze Terrain Detail
TerrainShading terrain,shade                    ; Setze Terrain Shading

RenderWorld                                     ; Rendere alle Objekte

If KeyHit(59) Then help=1-help

If Help Then
        Color 255,255,255
        Text 5,0,"TERRAINHEIGHT@CAMERA POSITION - "+Y#
        Text 5,15,"TERRAIN DETAIL @ "+detail
        If vmorph Then Text 5,25,"VERTEXMORPHING ENABLED" Else Text 5,25,"VERTEXMORPHING DISABLED"
        Text 5,35,"MODE "+mode+" EFFECT "+Effect#
        Text 5,Height-160,"CAMERA NAVIGATION (NUM PAD):"
        Text 25,Height-135,"Num8 - Lift Camera"
        Text 25,Height-120,"Num2 - Lower Camera"
        Text 25,Height-105,"Num4 - Rotate Left"
        Text 25,Height-90,"Num6 - Rotate Right"
        Text 25,Height-75,"Num1 - Move Left"
        Text 25,Height-60,"Num3 - Move Right  "
        Text 25,Height-45,"Num5 - Move Forward"
        Text 25,Height-30,"Num0 - Move Backward"
        Text Width-85,Height-10,"F1 TOGGLE THIS HELP SCREEN",1,1
        Color 0,255,0
        Text Width-150,0,"TERRAIN DEMO von DANIEL BACKMANN",1
        Text Width-150,10,"begleitend zur SEMESTERARBEIT CGA",1
        Text Width/2,Height-20,"CAMERA POSITION X:"+EntityX(Camera)+" Y:"+EntityY(Camera)+
        " Z:"+EntityZ(Camera),1
        Text Width/2,Height-40,"PRESS Num+ / Num- FOR SETTING LEVEL OF DETAIL",1
        Text Width/2,Height-50,"PRESS W FOR TOGGLE WIREFRAME MODE",1
        Text Width/2,Height-60,"PRESS V FOR TOGGLE VERTEXMORPHING",1
        Text Width/2,Height-70,"PRESS T FOR TOGGLE TERRAIN SHADING",1
        Text Width/2,Height-85,"PRESS BACKSPACE FOR MODIFY TERRAIN",1

End If

Flip                                            ; Switche den aktuellen Buffer (Doublebuffering)
Wend                                            ; Ende While Schleife

End                                             ; Ende Hauptprogramm
```

```
Function Modify(x%,y%,Effect#)

 ; Setz die Kamera direkt vor das Event
 PositionEntity camera,x*ScaleTXZ,40,y-50,1
 RotateEntity camera,0,0,0,1


 ; C Matrix                                                    Fuelle Matrix mit Daten

  C(0,0)=0  C(0,1)=0  C(0,2)=0  C(0,3)=1  C(0,4)=1  C(0,5)=1  C(0,6)=1  C(0,7)=1  C(0,8)=1  C(0,9)=1
  C(1,0)=0  C(1,1)=0  C(1,2)=1  C(1,3)=1  C(1,4)=1  C(1,5)=1  C(1,6)=1  C(1,7)=1  C(1,8)=1  C(1,9)=1
  C(2,0)=0  C(2,1)=1  C(2,2)=1  C(2,3)=1  C(2,4)=1  C(2,5)=1  C(2,6)=1  C(2,7)=1  C(2,8)=1  C(2,9)=1
  C(3,0)=1  C(3,1)=1  C(3,2)=1  C(3,3)=1  C(3,4)=1  C(3,5)=1  C(3,6)=1  C(3,7)=1  C(3,8)=1  C(3,9)=1
  C(4,0)=1  C(4,1)=1  C(4,2)=1  C(4,3)=1  C(4,4)=1  C(4,5)=1  C(4,6)=1  C(4,7)=1  C(4,8)=1  C(4,9)=1
  C(5,0)=1  C(5,1)=1  C(5,2)=1  C(5,3)=1  C(5,4)=1  C(5,5)=0  C(5,6)=0  C(5,7)=0  C(5,8)=0  C(5,9)=0
  C(6,0)=1  C(6,1)=1  C(6,2)=1  C(6,3)=1  C(6,4)=1  C(6,5)=0  C(6,6)=0  C(6,7)=0  C(6,8)=0  C(6,9)=0
  C(7,0)=1  C(7,1)=1  C(7,2)=1  C(7,3)=1  C(7,4)=1  C(7,5)=0  C(7,6)=0  C(7,7)=0  C(7,8)=0  C(7,9)=0
  C(8,0)=1  C(8,1)=1  C(8,2)=1  C(8,3)=1  C(8,4)=1  C(8,5)=0  C(8,6)=0  C(8,7)=0  C(8,8)=0  C(8,9)=0
  C(9,0)=1  C(9,1)=1  C(9,2)=1  C(9,3)=1  C(9,4)=1  C(9,5)=0  C(9,6)=0  C(9,7)=0  C(9,8)=0  C(9,9)=0
 C(10,0)=1 C(10,1)=1 C(10,2)=1 C(10,3)=1 C(10,4)=1 C(10,5)=0 C(10,6)=0 C(10,7)=0 C(10,8)=0 C(10,9)=0
 C(11,0)=1 C(11,1)=1 C(11,2)=1 C(11,3)=1 C(11,4)=1 C(11,5)=0 C(11,6)=0 C(11,7)=0 C(11,8)=0 C(11,9)=0
 C(12,0)=1 C(12,1)=1 C(12,2)=1 C(12,3)=1 C(12,4)=1 C(12,5)=1 C(12,6)=1 C(12,7)=1 C(12,8)=1 C(12,9)=1
 C(13,0)=1 C(13,1)=1 C(13,2)=1 C(13,3)=1 C(13,4)=1 C(13,5)=1 C(13,6)=1 C(13,7)=1 C(13,8)=1 C(13,9)=1
 C(14,0)=0 C(14,1)=1 C(14,2)=1 C(14,3)=1 C(14,4)=1 C(14,5)=1 C(14,6)=1 C(14,7)=1 C(14,8)=1 C(14,9)=1
 C(15,0)=0 C(15,1)=0 C(15,2)=1 C(15,3)=1 C(15,4)=1 C(15,5)=1 C(15,6)=1 C(15,7)=1 C(15,8)=1 C(15,9)=1
 C(16,0)=0 C(16,1)=0 C(16,2)=0 C(16,3)=1 C(16,4)=1 C(16,5)=1 C(16,6)=1 C(16,7)=1 C(16,8)=1 C(16,9)=1

 ; G Matrix

  G(0,0)=0  G(0,1)=0  G(0,2)=0  G(0,3)=1  G(0,4)=1  G(0,5)=1  G(0,6)=1  G(0,7)=1  G(0,8)=1  G(0,9)=1
  G(1,0)=0  G(1,1)=0  G(1,2)=1  G(1,3)=1  G(1,4)=1  G(1,5)=1  G(1,6)=1  G(1,7)=1  G(1,8)=1  G(1,9)=1
  G(2,0)=0  G(2,1)=1  G(2,2)=1  G(2,3)=1  G(2,4)=1  G(2,5)=1  G(2,6)=1  G(2,7)=1  G(2,8)=1  G(2,9)=1
  G(3,0)=1  G(3,1)=1  G(3,2)=1  G(3,3)=1  G(3,4)=1  G(3,5)=1  G(3,6)=1  G(3,7)=1  G(3,8)=1  G(3,9)=1
  G(4,0)=1  G(4,1)=1  G(4,2)=1  G(4,3)=1  G(4,4)=1  G(4,5)=1  G(4,6)=1  G(4,7)=1  G(4,8)=1  G(4,9)=1
  G(5,0)=1  G(5,1)=1  G(5,2)=1  G(5,3)=1  G(5,4)=1  G(5,5)=0  G(5,6)=0  G(5,7)=0  G(5,8)=0  G(5,9)=0
  G(6,0)=1  G(6,1)=1  G(6,2)=1  G(6,3)=1  G(6,4)=1  G(6,5)=0  G(6,6)=0  G(6,7)=0  G(6,8)=0  G(6,9)=0
  G(7,0)=1  G(7,1)=1  G(7,2)=1  G(7,3)=1  G(7,4)=1  G(7,5)=0  G(7,6)=0  G(7,7)=0  G(7,8)=0  G(7,9)=0
  G(8,0)=1  G(8,1)=1  G(8,2)=1  G(8,3)=1  G(8,4)=1  G(8,5)=0  G(8,6)=1  G(8,7)=1  G(8,8)=1  G(8,9)=1
  G(9,0)=1  G(9,1)=1  G(9,2)=1  G(9,3)=1  G(9,4)=1  G(9,5)=0  G(9,6)=1  G(9,7)=1  G(9,8)=1  G(9,9)=1
 G(10,0)=1 G(10,1)=1 G(10,2)=1 G(10,3)=1 G(10,4)=1 G(10,5)=0 G(10,6)=1 G(10,7)=1 G(10,8)=1 G(10,9)=1
 G(11,0)=1 G(11,1)=1 G(11,2)=1 G(11,3)=1 G(11,4)=1 G(11,5)=0 G(11,6)=1 G(11,7)=1 G(11,8)=1 G(11,9)=1
 G(12,0)=1 G(12,1)=1 G(12,2)=1 G(12,3)=1 G(12,4)=1 G(12,5)=1 G(12,6)=1 G(12,7)=1 G(12,8)=1 G(12,9)=1
 G(13,0)=1 G(13,1)=1 G(13,2)=1 G(13,3)=1 G(13,4)=1 G(13,5)=1 G(13,6)=1 G(13,7)=1 G(13,8)=1 G(13,9)=1
 G(14,0)=0 G(14,1)=1 G(14,2)=1 G(14,3)=1 G(14,4)=1 G(14,5)=1 G(14,6)=1 G(14,7)=1 G(14,8)=1 G(14,9)=1
 G(15,0)=0 G(15,1)=0 G(15,2)=1 G(15,3)=1 G(15,4)=1 G(15,5)=1 G(15,6)=1 G(15,7)=1 G(15,8)=1 G(15,9)=1
 G(16,0)=0 G(16,1)=0 G(16,2)=0 G(16,3)=1 G(16,4)=1 G(16,5)=1 G(16,6)=1 G(16,7)=1 G(16,8)=1 G(16,9)=1

 ; A Matrix

  A(0,0)=0  A(0,1)=0  A(0,2)=0  A(0,3)=1  A(0,4)=1  A(0,5)=1  A(0,6)=1  A(0,7)=0  A(0,8)=0  A(0,9)=0
  A(1,0)=0  A(1,1)=0  A(1,2)=1  A(1,3)=1  A(1,4)=1  A(1,5)=1  A(1,6)=1  A(1,7)=1  A(1,8)=0  A(1,9)=0
  A(2,0)=0  A(2,1)=1  A(2,2)=1  A(2,3)=1  A(2,4)=1  A(2,5)=1  A(2,6)=1  A(2,7)=1  A(2,8)=1  A(2,9)=0
  A(3,0)=1  A(3,1)=1  A(3,2)=1  A(3,3)=1  A(3,4)=1  A(3,5)=1  A(3,6)=1  A(3,7)=1  A(3,8)=1  A(3,9)=1
  A(4,0)=1  A(4,1)=1  A(4,2)=1  A(4,3)=1  A(4,4)=1  A(4,5)=1  A(4,6)=1  A(4,7)=1  A(4,8)=1  A(4,9)=1
  A(5,0)=1  A(5,1)=1  A(5,2)=1  A(5,3)=1  A(5,4)=0  A(5,5)=0  A(5,6)=1  A(5,7)=1  A(5,8)=1  A(5,9)=1
  A(6,0)=1  A(6,1)=1  A(6,2)=1  A(6,3)=1  A(6,4)=0  A(6,5)=0  A(6,6)=1  A(6,7)=1  A(6,8)=1  A(6,9)=1
  A(7,0)=1  A(7,1)=1  A(7,2)=1  A(7,3)=1  A(7,4)=0  A(7,5)=0  A(7,6)=1  A(7,7)=1  A(7,8)=1  A(7,9)=1
  A(8,0)=1  A(8,1)=1  A(8,2)=1  A(8,3)=1  A(8,4)=0  A(8,5)=0  A(8,6)=1  A(8,7)=1  A(8,8)=1  A(8,9)=1
  A(9,0)=1  A(9,1)=1  A(9,2)=1  A(9,3)=1  A(9,4)=1  A(9,5)=1  A(9,6)=1  A(9,7)=1  A(9,8)=1  A(9,9)=1
 A(10,0)=1 A(10,1)=1 A(10,2)=1 A(10,3)=1 A(10,4)=1 A(10,5)=1 A(10,6)=1 A(10,7)=1 A(10,8)=1 A(10,9)=1
 A(11,0)=1 A(11,1)=1 A(11,2)=1 A(11,3)=1 A(11,4)=1 A(11,5)=1 A(11,6)=1 A(11,7)=1 A(11,8)=1 A(11,9)=1
 A(12,0)=1 A(12,1)=1 A(12,2)=1 A(12,3)=1 A(12,4)=0 A(12,5)=0 A(12,6)=1 A(12,7)=1 A(12,8)=1 A(12,9)=1
 A(13,0)=1 A(13,1)=1 A(13,2)=1 A(13,3)=1 A(13,4)=0 A(13,5)=0 A(13,6)=1 A(13,7)=1 A(13,8)=1 A(13,9)=1
 A(14,0)=1 A(14,1)=1 A(14,2)=1 A(14,3)=1 A(14,4)=0 A(14,5)=0 A(14,6)=1 A(14,7)=1 A(14,8)=1 A(14,9)=1
 A(15,0)=1 A(15,1)=1 A(15,2)=1 A(15,3)=1 A(15,4)=0 A(15,5)=0 A(15,6)=1 A(15,7)=1 A(15,8)=1 A(15,9)=1
 A(16,0)=1 A(16,1)=1 A(16,2)=1 A(16,3)=1 A(16,4)=0 A(16,5)=0 A(16,6)=1 A(16,7)=1 A(16,8)=1 A(16,9)=1

 For i= 0 To 16
  For j= 0 To 9
    If C(i,j)=1 Then ModifyTerrain terrain,(x-16)+j,(y+8)-i,Effect,1      ; Vertex des Terrains an Position x,y
    If G(i,j)=1 Then ModifyTerrain terrain,(x-5)+j,(y+8)-i,Effect,1       ; auf den Wert Effekt setzen, Effekt
    If A(i,j)=1 Then ModifyTerrain terrain,(x+6)+j,(y+8)-i,Effect,1       ; kann 0 flach oder 1 hoch sein
  Next                                                                    ; abhängig von der Höhenskalierung des Terrains
 Next
End Function


Function SetWater()
```

```
	water=CreatePlane()                                ; Erstellen einer Fläche
	PositionEntity water,0,8,0                         ; Positionieren der Fläche

	wtex1=LoadTexture("WATER2.TGA",8)                  ; Wassertextur 1 laden
	wtex2=LoadTexture("WATER-2_MIP.BMP",8)             ; Wassertextur 2 laden
	ScaleTexture wtex1,64,64                           ; skalieren der Texturen
	ScaleTexture wtex2,15,15
	EntityTexture water,wtex1,0,1                      ; zuweisen der Texturen (siehe Terrain)
	EntityTexture water,wtex2,0,0
	TextureBlend wtex1,3                               ; Wassertextur 1 addieren
	EntityAlpha water,.7                               ; Setzen der Objekttransparenz auf 70%

End Function

Function InitializeCamera()

	camera=CreateCamera()                              ; Erstellen eines Kameraobjektes
	PositionEntity camera,462,25,50                    ; Positionieren der Kamera
	RotateEntity camera,0,45,0                         ; Rotation der Kamera 45° auf der Y-Achse
	CameraFogMode camera,1                             ; Linearer Horizontnebel AN
	CameraFogColor camera,128,196,255                  ; Nebelfarbe
	CameraFogRange camera,250,450                      ; Nebelentfernung Entfernung 0%, Entfernung 100%
	CameraClsColor camera,128,196,255                  ; Kamerahintergrundfarbe

End Function

Function InitializeTerrain()

	terrain=LoadTerrain("heightmap_2.bmp")             ; Erstellen eines Terrainobjektes
	TerrainDetail terrain,detail,vmorph                ; Terraindetail einstellen
	ScaleEntity terrain,ScaleTXZ,ScaleTY,ScaleTXZ      ; Terrain skalieren
	PositionEntity terrain,0,0,0                       ; Terrain auf 0 Punkt verschieben falls es nicht dort ist

   plane=CreatePlane(1)                                ; Horizont erstellen
	PositionEntity plane,0,-1,0                        ; Horizont etwas nach unten versetzen wegen
ueberlappender Faces

	basetex=LoadTexture("mossyground.bmp",8)           ; Grastextur laden
	texlvl2=LoadTexture("stone47.bmp",8)               ; Erdtextur laden
	texlvl1=LoadTexture("heightmap_2.tga",2)           ; Höhentextur laden
	;Animtex=LoadAnimTexture("powerup2.bmp",8,20,20,0,18)

	TextureBlend basetex,3                             ; Grastextur addieren
	TextureBlend texlvl2,2                             ; Erdtextur multiplizieren
	TextureBlend texlvl1,1                             ; Höhentextur Alphakanal
	;TextureBlend Animtex,3

	EntityTexture terrain,basetex,0,1                  ; Grastextur dem Terrain zuordnen (Index 1)
	EntityTexture terrain,texlvl2,0,0                  ; Erdtextur dem Terrain zuordnen (Index 0)
	EntityTexture terrain,texlvl1,0,2                  ; Höhentextur dem Terrain zuordnen (Index 2)
	;EntityTexture terrain,Animtex,0,3

	; Texturschichten:
	;
	; Textur Index 7 (8 Texturen sind bei Blitz3D Multitexturmaximum)
	;     /|\                        - Höhentextur
	;      |                         - Grastextur
	; Textur Index 0      - Erdtextur
	; Objekt/Entity


	ScaleTexture basetex,5,5                           ; Grastextur skalieren X*5, Z*5
	ScaleTexture texlvl2,15,15                         ; Erdtextur skalieren X*15, Z*15
	ScaleTexture texlvl1,512,512                       ; Höhentextur skalieren X*512, Z*512
	;ScaleTexture Animtex,512,512

	EntityTexture plane,basetex,0,1        ; Gras und Erdtexture auch dem Horizont zuweisen in selber Ordnung
	EntityTexture plane,texlvl2,0,0        ; damit es mit dem Modell identisch ist und sich nicht all zu sehr
	                                       ; davon abhebt.

End Function

Function CreateTree(x%,y%)

	otree=LoadSprite("tree-002.bmp",4)
	z=TerrainY(terrain,x,1,y)
	PositionEntity otree,x,z+8,y
```

```
        ScaleSprite otree,10,10

End Function
=====================================(END Terrain_DEMO.bb)=====================================
```

LOD.bb

```
Graphics3D 800,600,16,1
SetBuffer BackBuffer()

Dither True                                          ;Aktiviere gerasterte Farbverläufe bei 16 bit Farbtiefe

; Create Camera and set position and Color
camera=CreateCamera()
PositionEntity camera,256,1,1
CameraClsColor camera,128,196,255

; Create LightSource
light=CreateLight()
RotateEntity light,15,0,0

; Load terrain
terrain=LoadTerrain( "heightmap.bmp" )

detail=4000                                          ;Terrain Grunddetail auf 4000

; Set terrain detail, enable vertex morphing
TerrainDetail terrain,detail,True

; Scale terrain
ScaleEntity terrain,1,50,1

; Texture terrain
grass_tex=LoadTexture( "mossyground.bmp",8 )         ;Lade Mipmapped Grastextur
EntityTexture terrain,grass_tex,0,1                  ;Weise Textur dem Terrain zu
ScaleTexture grass_tex,2,2                           ;Skaliere Textur
TerrainShading terrain,True                          ;Aktiviere Geländeschattierung
; Wireframe OFF
wframe%=0

While Not KeyDown( 1 )

; W - Toggles wireframe
If KeyHit(17) wframe=1-wframe

WireFrame wframe

; Num +/- toggle Level of Detail
If KeyDown(78) Then detail=detail+10
If KeyDown(74) Then detail=detail-10

TerrainDetail terrain,detail,True

; CursorKeys move Camera
If KeyDown( 203 )=True Then x#=x#-0.10
If KeyDown( 205 )=True Then x#=x#+0.10
If KeyDown( 208 )=True Then y#=y#-0.10
If KeyDown( 200 )=True Then y#=y#+0.10
If KeyDown( 44 )=True Then z#=z#-0.10
If KeyDown( 30 )=True Then z#=z#+0.10

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.5
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.5

x#=EntityX(camera)
y#=EntityY(camera)
z#=EntityZ(camera)

; Locate MapHeight at Cameraposition
terra_y#=TerrainY(terrain,x#,y#,z#)+5

; Set Camera at Terrainlevel
PositionEntity camera,x#,terra_y#,z#
```

```
RenderWorld

Text 10,0,"Use cursor keys to move about the terrain / Num+/- to change Terrain Detail"
Text 10,10,"Use W Key to toggle Wireframe ON/OFF"
Text 10,20,"Use Cursor Keys to move Camera around"
Text 10,30,"Detail at "+detail%
Text 10,GraphicsHeight()-20,"Programmed by Daniel Backmann"
Flip

Wend

End

=========================================( Ende LOD.bb)=========================================
```

BSPTest.bb

```
;Test Programm für BSP Modelle

Graphics3D 800,600,16,1
SetBuffer BackBuffer()

; AntiAlias Einschalten / Funktioniert nicht wenn diese Funktion ueber
; den Treiber der Grafikkarte deaktiviert wurde.

AntiAlias True
Dither True                                 ;Aktiviere Farbverlaufsrasterung bei 16bit Farbtiefe
; Create camera
camera=CreateCamera()
CameraRange camera,1,5000                   ;Kamera Clipping Range von 1 bis 5000
PositionEntity camera,1,250,1               ;Positionieren der Kamera

; Create LightSource
light=CreateLight()                         ;Generiere Lichtquelle

; Load BSP Modell

BSP=LoadBSP("ctf_voy1.bsp",0.2)             ;Lade BSP Karte, Gammawert 0.2 (0 = Standard)
BSPAmbientLight BSP,64,64,64                ;Setze Ambientbeleuchtung des BSP Modells -> Dunkelgrau
Lighting=0                                  ;Beleuchtung VERTEX
Rotate=0                                    ;MeshRotation AUS

Arial=LoadFont("Arial",14,1,0,0)            ;Lade Arialfont, 14Punkte, Fett
SetFont Arial                               ;Setze Arial als aktiven Font


While Not KeyDown( 1 )

 ; BSP Rotation an/aus
 If KeyHit(19) Then Rotate=1-Rotate         ;Dient der Veranschaulichung das BSP Entities sich
 If Rotate=1 Then TurnEntity BSP,0,.5,0     ;weitestgehend wie gewoehnliche Entities verhalten

 ; Beleuchtungsmodell umstellen VERTEX/LIGHTMAP
 If KeyHit(57) Then Lighting=1-Lighting     ;Lighting 0 = VERTEX / 1 = LIGHTMAP
 BSPLighting BSP, Lighting

 ; Kamera Höhenjustierung
 If KeyDown(72) Then MoveEntity camera,0,2,0
 If KeyDown(80) Then MoveEntity camera,0,-2,0

 ; Kamera Winkeljustierung (oben/unten)
 ;If KeyDown(71) Then TurnEntity camera,1,0,0
 ;If KeyDown(73) Then TurnEntity camera,-1,0,0

 ; Kamera Rotation
 If KeyDown(75) Then TurnEntity camera,0,1,0
 If KeyDown(77) Then TurnEntity camera,0,-1,0

 ; Kamera Bewegungen
 If KeyDown(76) Then MoveEntity camera,0,0,2    ;Num5 -> bewege Kamera 2 Einheiten nach vorn
 If KeyDown(79) Then MoveEntity camera,-2,0,0   ;Num1 -> bewege Kamera 2 Einheiten nach links
 If KeyDown(81) Then MoveEntity camera,2,0,0    ;Num3 -> bewege Kamera 2 Einheiten nach rechts

        ;MoveEntity richtet seine X,Y,Z Achsen immer an der Ausrichtung des Objektes aus
        ;dadurch ist ein positiver Wert der Z Achse immer eine Bewegung des Objektes von sich aus nach vorne
```

```
RenderWorld

 Text 5,0,"MAP NAME: Capture The Flag _ Voyager 1 / Startrek Voyager: Elite Force"
 If Lighting Then Text 5,20,"LIGHTING MODELL:   Lightmap"  Else Text 5,15,"LIGHTING MODELL:   Vertex Lighting"
 Text 5,500,"NAVIGATE BY:"
 Text 5,520,"Num8 / Num2 - MOVE CAMERA UP/DOWN"
 Text 5,530,"Num4 / Num6 - TURN CAMERA LEFT/RIGHT"
 Text 5,540,"Num1 / Num3 - MOVE CAMERA LEFT/RIGHT"
 Text 5,550,"Num5            - MOVE CAMERA FORWARD"
 Text 5,570,"Press SPACEBAR to toggle LIGHTING MODELL"
 Text 5,580,"Press R to toggle BSP Rotation"
 Text 600,590,"Programmiert von Daniel Backmann",1,1
Flip
Wend

End

=======================================( ENDE BSPTest.bb )=======================================
```

Dynamic_Textures.bb

```
Graphics3D 800,600,16,1
SetBuffer BackBuffer()

Dither True                                     ;Rasterfarbverlauf fuer 16Bit Farbtiefe EIN

pivot=CreatePivot()                             ;Erzeuge Pivotpunkt
PositionEntity pivot,0,2,0


;Erzeuge Kamera 1
camera01=CreateCamera()
PositionEntity camera01,0,20,0

;Erzeuge Kamera 2
camera02=CreateCamera()
TurnEntity camera02,0,180,0                     ;Drehe Kamera um 180°
PositionEntity camera02,10,7,0                  ;Positioniere sie vor dem Quader
CameraViewport camera02,0,0,128,128             ;Begrenze den Ausschnitt auf die Texturgroesse
CameraClsColor camera02,128,128,128             ;KameraHintergrund Grau

texture=CreateTexture( 128,128 )                ;Erstelle leere Textur "texture"
fonttex=CreateTexture( 128,128 )                ;Erstelle leere Textur "fonttex"

atex=LoadAnimTexture("powerup2.bmp",13,20,20,0,18)   ;Lade Animierte Textur

SetBuffer TextureBuffer(fonttex)                ;Setze aktuellen Buffer auf Texturbuffer von fonttex
ClsColor 255,255,255                            ;Setze Loeschfarbe auf Weiss
Cls                                             ;Loesche den Buffer

font=LoadFont("arial",12)                       ;Lade den Font ARIAL 12Pt
SetFont font                                    ;Aktiviere Font
Color 0,0,0                                     ;Schriftfarbe schwarz
Text 0,0,"Objects in the mirror "
Text 0,12,"are closer than they appear."

SetBuffer BackBuffer()                          ;Setze Speicher zurueck auf Bildschirmspeicher

c1=CreateCube()                                 ;Erzeuge Wuerfel 1
PositionEntity c1,0,7,2                         ;positioniere Wuerfel 1
ScaleEntity c1,3.5,3.5,2                        ;Skaliere Wuerfel 1

cube=CreateCube()                               ;Erzeuge Wuerfel 2
PositionEntity cube,0,7,0                       ;positioniere Wuerfel 2
ScaleEntity cube,3,3,.1                         ;Skaliere Wuerfel 2

EntityTexture cube,texture,0,0                  ;Texturiere Wuerfel 2, texture Texturlevel0
EntityTexture cube,fonttex,0,1                  ;Texturiere Wuerfel 2, fonttex Texturlevel1

TextureBlend texture,1                          ;Texture Alphakanal
TextureBlend fonttex,2                          ;Fonttex Multiplizieren

light=CreateLight()                             ;Generiere Lichtquelle
TurnEntity light,45,45,0
```

```
grid_tex=CreateTexture( 16,16,8,1 )                ;Erzeuge leere Textur 16x16 Mippmapped, SingleFrame
ScaleTexture grid_tex,10,10                        ;Skaliere Texture
SetBuffer TextureBuffer( grid_tex )                ;Setz aktuellen Buffer auf Texturebuffer von grid_tex
ClsColor 255,255,255:Cls:ClsColor 0,0,0            ;Setz Loeschfarbe weiss, loesche buffer, setz loeschfarbe schwarz
Color 192,192,192:Rect 0,0,8,8:Rect 8,8,8,8        ;Setze Vordergrundfarbe hellgrau, zeichne 2 Rechtecke
SetBuffer BackBuffer()                             ;Setze Buffer zurueck auf Bildschirmbuffer

t_sphere=CreateSphere( 16,pivot )                  ;Erzeuge Kugel, 16 Segmente, pivot ist parent
ScaleEntity t_sphere,2,2,2                         ;Skaliere Kugel *2
EntityShininess t_sphere,.2                        ;Setze Glanzwert auf .2
MoveEntity t_sphere,0,0,30                         ;Verschiebe Kugel

plane=CreatePlane()                                ;Erzeuge Flaeche
EntityTexture plane,grid_tex                       ;Texturiere Flaeche mit Gridtextur

ScaleTexture texture,-1,1                          ;Spiegele texture an der y achse
ScaleTexture atex,.2,.2                            ;Skaliere die animierte Texture auf 1/5
d#=-20

f%=0                                               ;Framecount auf 0

While Not KeyHit(1)

        EntityTexture t_sphere,atex,f%             ;Aktuallisiere animierte Textur

        If KeyDown(30) d=d+1
        If KeyDown(44) d=d-1
        If KeyDown(203) TurnEntity camera01,0,-3,0
        If KeyDown(205) TurnEntity camera01,0,+3,0

        PositionEntity camera01,0,7,0
        MoveEntity camera01,0,0,d

        ;TurnEntity cube,.1,.2,.3
        TurnEntity pivot,0,1,0                     ;drehe Pivot -> dadurch rotiert Kugel um parent (pivot)

        UpdateWorld

        f%=f%+1                                    ;zaehle Framecount eins hoch
        If f%>17 f%=0                              ;wenn frame 17 erreicht setze framecount wieder auf 0

        ;If KeyHit( 57 )=True Then cam=1-cam

        ;If cam=0 Then
                HideEntity camera01                ;Versteck Kamera 1
                ShowEntity camera02                ;Zeige Kamera 2

        RenderWorld
        CopyRect 0,0,128,128,0,0,0,TextureBuffer( texture ) ;kopiere rechteckige Auswahl in Texturbuffer von texture

        ;Else
                ShowEntity camera01                ;Zeige Kamera 1
                HideEntity camera02                ;Versteck Kamera 2
        ;EndIf

        RenderWorld
        Text 5,0,"PRESS ESC TO QUIT"
        Text 5,20,"PRESS CURSOR LEFT/RIGHT TO ROTATE CAMERA"
        Text 5,30,"PRESS A / Y TO ZOOM IN OR OUT"
   Color 0,0,0
   Text 550,590,"Programmiert von Daniel Backmann",1,1
   Color 192,192,192
        Flip
Wend

End

==================================( ENDE Dynamic_Textures.bb )==================================
```

Quellenverzeichnis:

- Flipcode - GameDevelopment / Quake 2 BSP File Tutorial (http://www.flipcode.com)
- BlitzBase OnlineHilfe (http://www.blitzbase.de)
- BlitzBasic Forum/Homepage (http://www.blitzbasic.com)
- CFXWeb (http://www.cfxweb.net/~cfxamir/index.html)
- Uni Tübingen (http://www.gris.uni-tuebingen.de/gris/grdev/java/applets/bsptrees/html/index.html)