View-Dependent Refinement of Progressive Meshes

Hugues Hoppe, Microsoft Research

Seminararbeit von Thomas Giger

Inhaltsverzeichnis

1 Einführung

In der Computergrafik werden häufig Objekte mittels Dreiecksvermaschungen dargestellt. Damit diese Objekte beim nahen Betrachten nicht zu eckig wirken, bestehen die Vermaschungen oft aus sehr vielen, kleinen Dreiecken. Dies hat allerdings auch negative Auswirkungen. Wenn man ein Objekt in der Szene nur aus weiter Ferne betrachtet, das heisst, wenn das Objekt auf dem Bildschirm nur sehr klein erscheint, braucht man diese hohe Auflösung nicht. Die vielen Dreiecke belasten nur die Rechengeschwindigkeit. Oft werden daher die Vermaschungen in verschiedenen Auflösungen (Levels of Detail LOD) im Speicher gehalten. Je nach Abstand des Objektes vom Beobachter wird mit der optimalen Version gearbeitet. Diese Objekte sind oft vom Originalobjekt her sichtunabhängig verfeinert worden. Es sind also immer dieselben Geometriedaten im Objekt enthalten, egal aus welcher Richtung man es betrachtet.

View-Dependent Refinement Unter sichtabhängigem Verfeinern versteht man eine Verfeinerung derjenigen Dreiecke, die man auf dem Bildschirm tatsächlich sehen kann. Die restlichen Bereiche sind nur grob unterteilt. In seinem Ansatz verzichtet Hugues Hoppe auf die Speicherung von verschieden aufgelösten Objekten und verfeinert ein Objekt immer von seinem Originalmesh her.

2 Progressive Meshes

Die Progressive Meshes (PM) bilden die Grundlage für die neue Methode. Der Autor hat PM in einem vorangehenden Paper beschrieben.

2.1 Konstruktion eines PM

Jedes PM wird in einem Preprocessing-Schritt aus einem beliebigen Originalmesh konstruiert. Dabei spielen die folgenden Operationen eine wichtige Rolle:

2.1.1 Edge Collapse (ecol)

Die ecol-Operation zieht eine Kante in sich zusammen. Ein neuer Vertex entsteht aus der zusammengezogenen Kante und deren beiden Endvertices. Im allgemeinen Fall verschwinden auch die zwei an der Kante anliegenden Dreiecke. Ein ecol vergröbert also ein Mesh.

2.1.2 Vertex Split (vsplit)

vsplit ist die inverse Operation zu ecol; mit vsplits wird ein Mesh verfeinert.

2.1.3 Vorgehen

Ein PM wird mit einer Energiefunktion berechnet. Das Originalmesh M bildet das Energieminimum. Je mehr sich ein Mesh vom Original entfernt, desto höher wird dessen Energie. Ein ecol/vsplit enspricht daher einer Energiedifferenz ΔE. Zunächst wird diejenige Kante entfernt, deren ecol das kleinste ΔE hat. Gleichzeitig wird die Information, die vsplit braucht, um den ecol wieder rückgängig zu machen, in den Speicher geschrieben. Nun haben wir ein einfacheres Mesh sowie die Information zur Rekonstruktion des Originalmeshes. Darauf folgt der nächste Vereinfachungsschritt. Wieder wird die Kante mit dem kleinsten ΔE gesucht und zwar im Mesh, so wie es sich zurzeit präsentiert. Wie vorher wird der ecol ausgeführt und die vsplit-Information festgehalten. Wir verfeinern beliebig weit. Im Normalfall bis auf 5-100 übrigbleibende Dreiecke. Dieses Mesh nennt man Basismesh M_0. n sei die Anzahl der durchgeführten Schritte. Dann sind im Speicher zusätzlich auch n vsplit Informationen. M_0 und die vsplits bilden nun das PM.

$$PM = \{M_0, vsplit_0, vsplit_1, .., vsplit_n\}$$

Damit kann man ein Objekt in beliebiger (aber noch sichtunabhängiger) Auflösung darstellen, je nachdem wieviele vsplits man von M_0 her ausführt.

Die PM-Konstruktion ist eine lokale Optimierung. Wir haben nun nicht das beste Mesh, das sich mit n ecols realisieren lässt. Denn wir haben in jedem Schritt vorausgesetzt, dass am bisher generierten Mesh nichts mehr geändert wird. Dieser Gedanke ist grundlegend. Eine globale Optimierung würde es nicht erlauben, von Mesh M_x zu Mesh M_{x-1} durch die Ausführung eines ecols zu gelangen. Unter Umständen müssten sehr viele vsplits und ecols durchgeführt werden.

2.2 Anwendungen von PM

Mesh simplification Wie vorhin aufgezeigt, lässt sich ein Mesh beliebig weit vereinfachen.

Level of Detail approximation Ein Objekt kann in verschiedenen LOD-Stufen abgespeichert werden. Zur Laufzeit wird dann zwischen den Stufen ausgewählt.

Progressive transmission Wenn man ein Objekt auf einem Kommunikationskanal überträgt, so wird zunächst M_0 über den Kanal geschickt. Dann folgen die vsplits, natürlich in abnehmender "Wichtigkeit". Der Empfänger bricht die Übertragung dann ab, wenn er mit der Genauigkeit zufrieden ist.

Mesh compression Ein Mesh wird komprimiert, indem man die PM-Datenstruktur an einer beliebigen Stelle abschneidet.

Selective refinement PMs bilden die Basis für die im Verlauf der Arbeit beschriebenen Selective Refinement Techniken.

3 Das Selective Refinement Framework

Das Selective Refinement Framework bildet die Grundlage, um irgend eine bestimmte Region auf einem Mesh zu verfeinern oder zu vergröbern.

3.1 Die Definiton der Basisfunktion

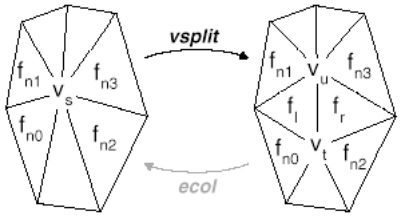

Abbildung 1: Edge Collapse und Vertex Split

Im Gegensatz zum vorangehenden Paper wurden die Definitionen von ecol und vsplit ein wenig verändert. Die geometrische Bedeutung bleibt allerdings dieselbe.

$$ecol(v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3})$$

$$vsplit(v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3})$$

Der Vertex v_s verschwindet beim vsplit, und zwei neue Vertices werden generiert. Die Flächen $f_{n0..3}$ ändern zwar ihre Form, bleiben aber erhalten.

Vorbedingungen Die Vorbedingungen für einen vsplit sind:

- v_s muss aktiv sein (im aktuellen Mesh vorhanden).
- $f_{n0..3}$ müssen ebenfalls aktiv sein.

Die Vorbedingungen für ecol sind:

- v_t und v_u müssen aktiv sein.
- Die Flächen müssen so angeordnet sein wie in Abbildung 1. Das heisst, es darf beispielsweise kein zusätzliches Dreieck bestehen zwischen f_l und f_{n0}, was durchaus möglich wäre.

Eigenschaften Diese Vorbedingungen garantieren unter anderem folgende Eigenschaften:

- Sind die Vorbedingungen für vsplit erfüllt, dann ist die Konfiguration der Dreiecke wie in Bild 1.
- Nachdem ein bestimmter vsplit durchgeführt wurde, existieren die neugeschaffenen Flächen in allen Meshes, bis der zugehörige ecol diese Flächen selber löscht.

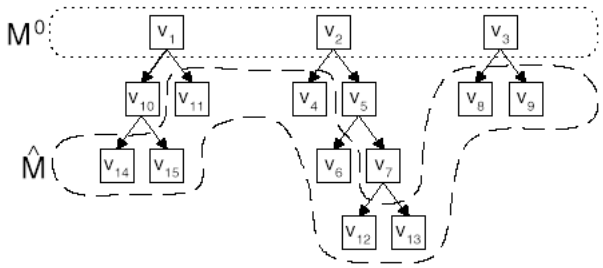

Abbildung 2: Die Vertex Hierarchy

3.2 Vertex hierarchy

Wenn man nun ein PM in den Speicher lädt, so wird eine Datenstruktur wie in Abbildung 2 angelegt. Die Vertices aus M_0 bilden die Wurzeln der Bäume. Dann werden für jeden vsplit-Datensatz zwei neue Vertices eingefügt. Am Schluss bilden die Originalvertices die Blätter des Baumes. Jedes beliebige Mesh, das gemäss den obigen Vorbedingungen kreiert wurde, kann man als Vertex Front in diese Hierarchy einzeichnen. Wird ein vsplit durchgeführt, so wandert die Front nach unten, bei ecols geht es aufwärts.

4 Die Kriterien zur Verfeinerung

Der vorhergehende Abschnitt lieferte die Werkzeuge, um ein Mesh selektiv zu verfeinern. Jetzt folgen die Kriterien, um ein Mesh so zu verfeinern, dass es für den Blickwinkel des Betrachters optimiert wird. Dazu benutzt Hoppe drei Kriterien.

4.1 Sichtbarer Bereich

Wenn sich ein Objekt im sichtbaren Bereich der Szene befindet, so soll es originalgetreu belassen werden. Befindet sich das Objekt oder ein Teil des Objekts ausserhalb des sichtbaren Bereiches, soll es möglichst vergröbert werden. Das wird folgendermassen sichergestellt:

- Die Nachbarschaft eines Vertex ist definiert als alle anliegenden Dreiecke und deren Vertices. Nun wird in einem ersten Schritt für jeden Vertex von $\hat{M}$ eine umhüllende Kugel um dessen Nachbarschaftsregion gelegt. Man speichert deren Radius in den Vertexdaten. Diese Berechnung erfolgt im Preprocessing.

- Bis zu diesem Punkt haben nur die Vertices aus dem Originalmesh einen Radius. Beim Ladevorgang erhalten auch die Vertices aus dem oberen Teil der Hierarchy eine Umkugel. Wie vorhin soll diese Kugel die von vsplits beeinflusste Region einhüllen. Um die Kugel zu berechnen, legt man einfach eine Kugel um die beiden Kugeln der Kinder-Vertices. Das Zentrum der neuen Kugel soll der besagte Vertex bilden. Der so erhaltene neue Radius wird wiederum in den Vertex-Daten festgehalten. Je höher ein Punkt in der Hierarchy, desto grösser ist seine Kugel.

- Zur Laufzeit testen wir nur noch, ob die Kugel eines Vertex den sichtbaren Bereich der Szene schneidet. Der sichtbare Bereich ist durch seine vier Begrenzungsebenen definiert. Man benutzt folgendes einfache Kriterium:

$$a_i v_x + b_i v_y + c_i v_z + d_i < -r_v, i = 1..4$$

4.2 Orientierung der Fläche

Auch vom Betrachter abgewandte Dreicke müssen nicht hoch aufgelöst sein. Dafür sorgt das zweite Kriterium:

- In jedem Vertex ist eine Normale gespeichert. Damit man eine Begrenzung für die Normalen der angrenzenden Dreiecke hat, betrachtet man

für jeden Vertex des Originalmeshes auch noch die Normalen der benachbarten Vertices. Man stelle sich nun vor, diese Normalen wären alle in einem Punkt festgebunden. Es entsteht ein Strauss von Normalen. Darum kann man einen Kegel legen, die die Normalen begrenzt. Die Berechnung der Kegel geschieht zur Ladezeit wie folgt. Man betrachtet eine Einheitskugel und trägt alle Normalen vom Ursprung her ab. Diese schneiden die Oberfläche in verschiedenen Punkten. Um diese Punkte legt man eine umhüllende Kugel S_{vr}. Aus der Kugel kann man auf einfache Weise den Kegel für den betreffenden Vertex berechnen. Weil schon die Normale im Vertex gespeichert ist, muss zusätzlich nur der Öffnungswinkel des Kegels gespeichert werden.

- Für die höheren Vertices der Hierarchy muss ein Kegel gefunden werden, der die Kegel der Kindervertices umgibt. Das geschieht, in dem man eine neue Kugel um die beiden Kugeln S_{vt} und S_{vu} der Kindervertices legt. Auch daraus wird wiederum der Öffnungswinkel berechnet. Der Öffnungswinkel wird beschränkt; der Kegel darf nicht mehr als den halben Raum aufspannen.

- Zur Laufzeit werten wir folgendes Kriterium aus: $(\mathbf{v} - \mathbf{e})$ ist dabei die Richtung, in die der Benutzer schaut, α_v ist der Öffnungshalbwinkel.

$$(\mathbf{v} - \mathbf{e}) \cdot \mathbf{n_v} > 0$$

$$and$$

$$((\mathbf{v} - \mathbf{e}) \cdot \mathbf{n_v})^2 > \|v - e\|^2 * sin^2\alpha_v$$

  Im Abbildung 3 sieht man den zweidimensionalen Fall. In 3D entsprechen die obigen Flächen dem Normalen- und dem Backfacing-Kegel (den Raum, indem man die am besagten Vertex anliegenden Flächen und deren zugehörige Region aus dem Originalmesh nicht sehen kann) Das obige Kriterium prüft, ob der Betrachter sich in der backfacing Region befindet. Dabei wird zur Vereinfachung eine Parallelprojektion angenommen.

4.3 Der Screen Space Error

Das Screen Space Error Kriterium sorgt für folgendes: Wenn ein Punkt des Originalmesh gerendert würde, so würde er in der Bildschirmprojektion auf den Bildpunkt a fallen. Weil er aber mit einem gröberen Mesh gerendert wird, fällt er auf Punkt b. Nun begrenzt man künstlich die Abweichung zwischen den beiden Punkten, indem man einen Toleranzwert angibt. Wird der

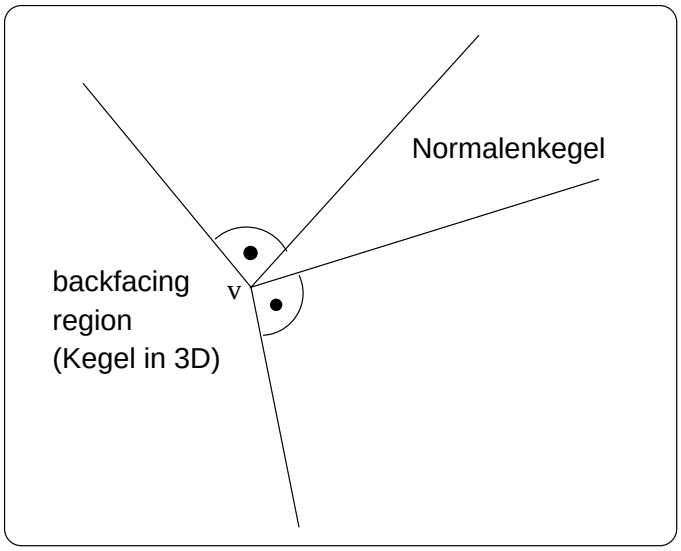

Abbildung 3: Normalenkegel und Backfacing Region

Wert überschritten, führt man einen vsplit durch. Diese Idee wird wie folgt realisiert:

- Beim Preprocessing wird für jeden Vertex zwei Werte μ und δ bestimmt, so dass sie einen Abweichungsraum wie in Abbildung 4 aufspannen. Sei N_v die Nachbarschaft eines Vertex. Sei $\hat{N}_v$ die entsprechende Region im Originalmesh. Setzt man an einen beliebigen Punkt in $\hat{N}_v$ die obige Form an, so soll auch ein Punkt aus N_v darin enthalten sein und umgekehrt.

- μ sorgt dafür, dass Objekte, die weit vom Benutzer entfernt sind, weniger verfeinert werden. Falls man die von μ_v aufspannte Kugel rendern würde, dann wäre die Kugel im Bild mehr oder weniger Pixel breit, abhängig von der Distanz des Objektes vom Benutzer. Ist die Kugel grösser als die erlaubte Tolreranz, so wird gesplittet.

- Flächen, deren Normalen auf den Benutzer zeigen, sollen weniger stark verfeinert werden. Der Benutzer kann den Fehler weniger gut wahrnehmen. Im Gegensatz dazu sollen Dreiecke, deren Normale mit der Blickrichtung einen rechten Winkel bilden, stärker verfeinert werden, weil der Betrachter Oberflächenunregelmässigkeiten stärker wahrnimmt, wenn er eine Fläche von der Seite, also als Silhouette sieht. Deshalb ist auch eine normale Komponente δ bei der Definition des Abweichungsraumes mitberücksichtigt worden. Würde man nur mit μ, also einem

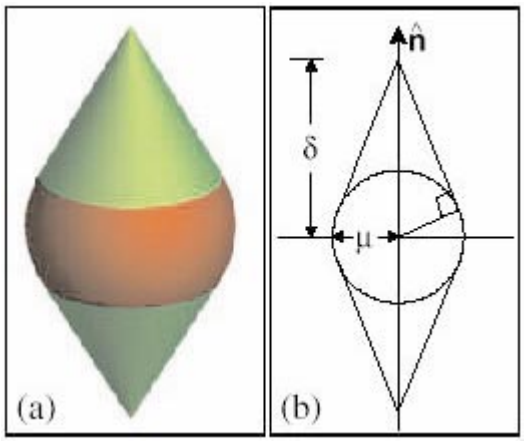

Abbildung 4: Der Abweichungsraum

kugelförmigen Abweichungsraum, arbeiten, so bekäme man ein viel grösseres μ und könnte den eben beschriebenen Effekt nicht ausnutzen.

- Zur Laufzeit wertet man die folgenden Kriterien aus:

$$\mu_v^2 \geq \kappa^2 \|\mathbf{v} - \mathbf{e}\|^2$$

or

$$\delta^2 * (\|\mathbf{v} - \mathbf{e}\|^2 - ((\mathbf{v} - \mathbf{e}) \cdot \mathbf{n_v})^2) \geq \kappa^2 \|\mathbf{v} - \mathbf{e}\|^4$$

Der erste Ausdruck testet das μ-Kriterium. $\mathbf{v} - \mathbf{e}$ ist die Blickrichtung. Im κ steckt die Toleranz drin. In der unteren Formel sieht man, dass der Winkel zwischen Vertexnormale und Blickrichtung betrachtet wird. Ist dieser kleiner, so kann δ grösser sein, ohne das ein vsplit ausgelöst wird.

5 Algorithmus

Der Algorithmus kämmt für jedes Frame die Liste der aktiven Vertices (Vertex Front, siehe Vertex Hierarchy) durch. Bei jedem einzelnen Vertex werden die Kriterien ausgewertet und je nachdem ein vsplit oder ein ecol ausgeführt. Die Vertex Front verschiebt sich dabei. Wenn die Kriterien einen ecol empfehlen, so wird dieser nur durchgeführt, wenn es die Vorbedingungen erlauben. Ganz im Gegensatz dazu werden vsplits immer durchgeführt. Wenn die Vorbedingungen nicht im vornherein erfüllt sind, führt der Algorithmus vsplits durch, bis die Bedingungen den beabsichtigten vsplit erlauben.

Laufzeit Die Laufzeit des Algorithmus beträgt (um eine Frame zu generieren) $O(|V_A| + |V_B|)$, das heisst die Anzahl der Vertices von Ausgangsmesh A plus diejenige von Endmesh B. Dies ist allerdings selten der Fall, nur wenn die beiden Ansichten völlig verschieden sind. Dabei würde mit ecols bis zu M_0 zurückgerechnet und von dort her wieder mit vsplits zu V_B. Im Normalfall sind die Ansichten in zwei aufeinanderfolgenden Frames ja ähnlich, und der Rechenaufwand ist viel kleiner.

Regulation der Framerate Die zeitliche Aufwand zur Berechnung eines Frames kann je nach Ansicht (Anzahl Dreiecke) sehr unterschiedlich sein. Deshalb kann die Toleranz des Screen Space Error angepasst werden. Steigt die Dreickzahl, lässt man auch die Toleranz steigen. Auf diese Weise erhält man konstantere Frameraten.

Triangle Strips Die meisten Grafiksysteme verlangen Dreicke als Triangle Strips. Da in unserem Fall die Anordnung und die Nachbarschftsbeziehungen dynamisch ändern, müssen die Strips in jedem Frame neu definiert werden. Ein Greedy Algorithmus löst dieses Problem. Bei einem Dreieck wird begonnen und die Nachbardreiecke werden dann hinzugefügt, wenn sie noch in keinem Strip sind. Ein Strip ist fertig, wenn der Algorithmus in einer Sackgasse steckt. Sucht man ein Nachbardreieck, so nimmt man, falls möglich, diejenigen Dreicke, die den Strip als Spirale im Uhrzeigersinn ausehen lassen. Dies reduziert Fragmentierungseffekte. Zusätzlich werden nur Nachbarn betrachtet, die aus dem selben Material sind. Damit vermeidet man unnötige Grafikstatuswechsel.

6 Resultate

6.1 Zeit

Der folgende Test lief auf einer SGI Indigo Extreme (150 MHz R4400, 128 MB). Als Testobjekt wurde ein Terrain vom Grand Canyon verwendet mit 600x600 vertices. Die PM-Konstruktion (inkl. Berechnung von μ_v und δ_v dauerte 10 Stunden, es resultierten 700'000 vertices. Die PM-Darstellung wurde auf 400'000 Dreiecke abgeschnitten. Die Ladezeit dauerte noch eine Minute (Berechnung von α_v und r_v). Zur Laufzeit wurde ein Framerate von 7.2 frames/s erreicht. Dabei war die CPU mit der Berechnung der Kriterien, der vsplits und ecols und der Generierung der Triangle strips nicht voll ausgelastet. Das Grafiksystem diktierte die Geschwindigkeit. Auf einem Impact Grafiksystem wurden 14 frame/s erreicht.

6.2 Speicher

Der Speicherbedarf beträgt $O(|V|)$. Pro Vertex des Originalmeshes fällt im Schnitt noch ein zweiter Vertex und 2 Faces an. Der Speicheraufwand pro Vertex im Originalmesh beträgt in Hoppe's Implementation etwa 224 Bytes. Der Autor schätzt, dass diese Zahl in einer optimierten Variante auf etwa 140 Bytes gesenkt werden könnte.

7 Ausblick

Zukünftige Forschungsgebiete, die an diese Arbeit anlehnen, sind u.a.:

- Memory management für grosse (Gelände-) Modelle
- Generierung von Geomorphs zur Laufzeit (Morphings zwischen zwei verschieden verfeinerten Meshes)
- Adaptive Verfeinerung von animierten Modellen
- Selektives Verfeinern zur Kollisionsdetektion

Literatur

[1] Hugues Hoppe. Progressive meshes. In *Computer Graphics (SIGGRAPH '96)*, pages 99–108, 1996.

[2] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Computer Graphics (SIGGRAPH '97)*, pages 199–198, 1997.

[3] Martin Knecht. Progressive meshes. GDV-Fachseminar '97, www.inf.ethz.ch/department/IS/cg/, 1997.