Seminararbeit
Progressive Forest Split Compression *PFS*

Philipp Kramer

26. Mai 1999

1 Einleitung

Das bearbeitete Paper [1] präsentiert einen Algorithmus zum Komprimieren von geometrischen Objekten. Die Komprimierung ist so gewählt, dass beim Entkomprimieren das Objekt stufenweise detailierter dargestellt werden kann. Da das Paper auf der Methode der *topological surgery* [2] aufbaut, ist deren Verständnis unumgänglich.

Im folgenden Text verwende ich die 'wir'-Form, womit natürlich die Autoren von [1] gemeint sind.

Die Objekte sind als Dreiecks*meshes* aufgebaut. Im folgenden Text werden wir immer annehmen, dass die Objekte keine Ränder haben, d.h. geschlossen sind.

2 Topological Surgery (*TS*)

Topological Surgery *TS* ist eine Technik zur Beschreibung von geometrischen Objekten, deren Oberfläche trianguliert ist. Sie versucht die inhärente Redundanz, die viele beliebte Darstellungen besitzen, zu minimieren. Bei der Komprimierung erhält sie die Konnektivität des *mesh*. Bei einer späteren Dekomprimierung müssen immer alle Knotenkoordinaten zugreifbar sein.

Einfache Polygone Als ein einfaches Polygon definieren wir ein Polygon, welches ein wohldefiniertes Inneres, keine sich schneidenden Kanten und keine inneren Knoten besitzt. Im folgenden Text werden wir auch triangulierte einfache Polygone, einfache Polygone nennen.

2.1 3D Objekte und Bäume

Wir können jedes triangulierte Objekt in ein einfaches Polygon (Abb. 2) verwandeln. Wenn wir das *mesh* an den Kanten E_P des Baumes, den sog. *cut edges* (Abb. 1 und 3) aufschneiden, dann können wir es auseinander ziehen und in der Ebene darstellen. Die Knoten V_P und Kanten E_P des triangulierten Polygons bilden einen Graphen G_P, in welchem wir einen spannenden

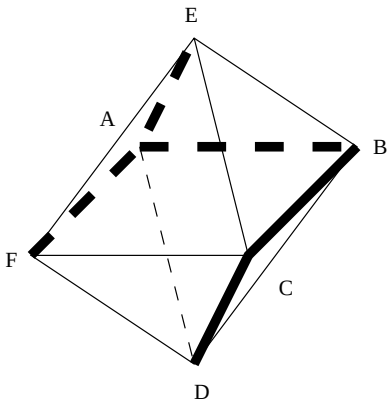

Abbildung 1: Objekt als 3D-*mesh*. Die *cut edges* sind mit starken Linien gezeichnet.

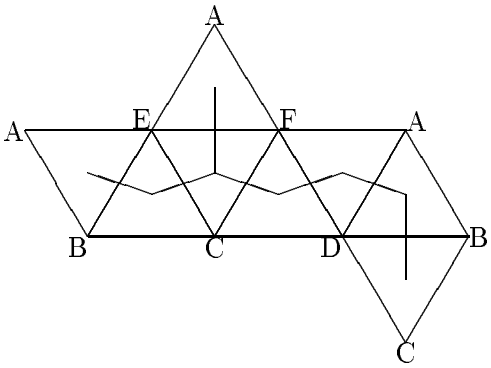

Abbildung 2: Dualer Graph der Trinagulierung des einfachen Polygons.

Baum markieren können. Die *cut edges* werden also bei dieser Operation verdoppelt und umranden das neu entstandene Polygon P. Wenn wir die Dreiecke von P durch Knoten V_D und die Nachbarsbeziehung von Dreiecken durch Kanten E_D ersetzen, dann erhalten wir dessen dualen Graphen G_D (Abb. 2).

- G_D hat ebenfalls die Gestalt eines Baumes.
- Da die Knoten des Baumes Dreiecke repräsentieren, ist G_D binär.
- Kanten aus V_P im Inneren des einfachen Polygons heissen *marching edges*.
- Kanten aus V_P am Rand des Polygons heissen *boundary edges*.

2.2 Komprimierte Darstellung von Bäumen

Eine Folge von Knoten, die jeweils mit genau einer Kante verbunden sind, heisst *run*. Einen Baum kann man sich aus *runs* und Verzweigungsknoten

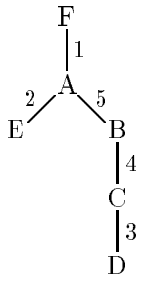

Abbildung 3: Baum der *cut edges*. Die Numerierung der Kanten geben die Reihenfolge der Kantenzusammenlegung bei der Rekonstruktion an.

aufgebaut vorstellen. Die Verzweigungs- und Blattknoten eines Baumes werden also durch *runs* verbunden. Wir traversieren den Baum in *pre-order*. Ein allgemeiner Baum kann damit als eine Folge von Tripeln dargestellt werden. Jedes Tripel bezieht sich auf einen *run* und besteht aus folgenden 3 Angaben:

- Die Länge ist definiert als die Anzahl Kanten des *runs*.
- Das *branching bit* zeigt an, ob der *run* einen Bruder hat (ob der Vaterknoten noch einen weiteren Sohn hat). Im Falle eines binären Baumes kann das *branching bit* weggelassen werden, da alle Verzweigungsknoten zwei Söhne besitzen.
- Das *leaf bit* schliesslich ist gesetzt, wenn der *run* in einem Blatt endet.

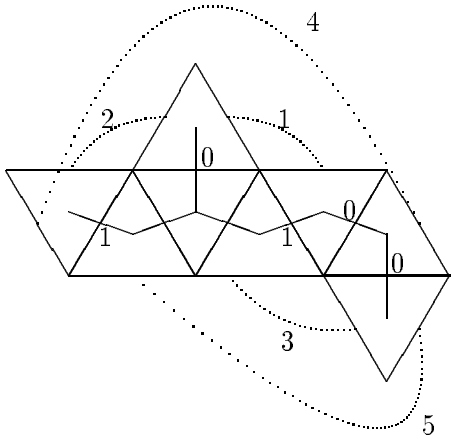

Abbildung 4: Kodierung der *marching edges* mit 0 und 1. Die gepunkteten Bogen markieren die Kanten die bei der Rekonstruktion zusammengefaltet werden müssen; diese lassen sich aus dem Baum der *cut edges* ableiten (Abb. 3).

2.3 Darstellung

Mit Hilfe der Technik aus Abschnitt 2.2 können wir die Topologie von Bäumen effizient kodieren. Im Abschnitt 2.1 haben wir gelernt 3D-Objekte als Bäume darzustellen. Es geht nun darum, eine Datenstruktur zu definieren, welche die gesamte Geometrie eines dreidimensionalen Objektes erfasst.

Das Objekt ist gegeben durch Knoten mit den Indizes von 0 bis $V-1$ und Dreiecke von 0 bis $T-1$. Knoten sind gegeben durch ihre drei Koordinaten, die Dreiecke durch drei Indizes in die Knotenmenge. Eine komprimierte Darstellung eines Modells ist gegeben durch:

VTREE Ein Feld von Tripeln, die den spannenden Baum der *cut edges* repräsentieren (siehe Abschnitt 2.2).

VCOR Was uns natürlich noch fehlt, sind die Koordinaten der Knoten. Ist die Distanz der Knoten im Knotenbaum gering, so sind auch die entsprechenden Knoten auf dem Objekt nahe beieinander gelegen. Deshalb können wir die Vorgänger im Knotenbaum verwenden, um die Position der Knoten des Objekts vorher zu sagen. Dazu werden bei der Kompression die Parameter einer Vorhersagefunktion auf das vorliegende Modell optimiert. Den Fehler der Vorhersage beheben wir mit einem korrektiven Term. Die korrektiven Terme werden nach ihrer Entropie komprimiert. Diese müssen mit jedem Knoten assoziiert werden. Wie wir in Abschnitt 2.2 gesehen haben, traversieren wir den Baum der *cut edges* in *pre-order*. Diese implizite Zuordnung der komprimierten Koordinatenkorrekturen zu den Objektknoten nützen wir aus, indem wir die Terme in der selben Reihenfolge abspeichern.

TTREE Ein Feld von Tripeln, die den spannenden Baum der Dreiecke repräsentiert (siehe Abschnitt 2.2). Da der Baum binär ist, kann das *branching bit* weggelassen werden.

MARCH Eine Folge von Bits, genannt *marching pattern*, welche die Anordnung der Dreiecke in den *runs* beschreibt. Diese Anordnung ist in *TTREE* noch nicht enthalten. Die *marching edges* sind diejenigen Kanten, welche je einen Knoten auf beiden Seiten des *runs* haben (vgl. Abschnitt 2.1). Wie das engl. Wort *march* schon andeutet, geht es darum, in welcher Richtung wir auf einem Dreiecks*run* marschieren. Jede *marching edge* hat einen Knoten mit ihrer Vorgängerin gemeinsam. Dieser gemeinsame Knoten kann entweder auf der linken oder rechten Begrenzung liegen; dazu brauchen wir ein Bit. Diese Bits werden in der Reihenfolge, wie sie vom Dekompressionsalgorithmus besucht werden, abgespeichert.

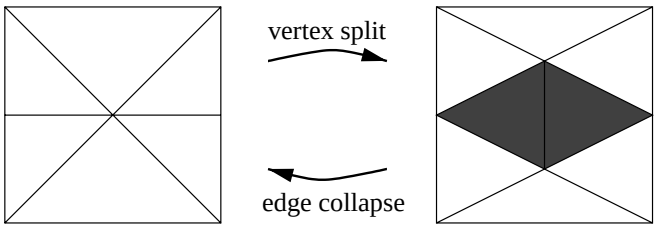

Abbildung 5: Die Graphik zeigt die beiden Grundoperationen des *PM* Verfahrens.

3 Progressive Forest Split (*PFS*)

3.1 Komprimierung von Geometrie

Während *single resolution* Verfahren nur dazu benutzt werden können, Diskspeicher und Übertragungsbandbreite zu sparen, ist es zusätzlich wünschenswert, ein Modell in einer fortlaufenden Art zu senden. Dies erlaubt das Objekt zu rendern, wenn noch gar nicht alle Daten zur Verfügung stehen. Ein ein Objekt ist im *progressive forest split* Verfahren mit einem *multiresolution mesh* dargestellt, und ist zusammengesetzt aus einem niedrig aufgelösten Dreiecks*mesh* und einer Folge von Verfeinerungsoperationen. Das *PFS* Schema unterliegt dabei einem Tradeoff zwischen Granularität und Kompressionsrate. Die höchste Kompression erhält man, indem die Anzahl Verfeinerungsschritte minimalisiert wird.

Viele Verfahren arbeiten nur für *meshes* mit rekursiver Struktur. Diese Einschränkung ist aber häufig nicht tragbar. Im Gegensatz dazu arbeitet *PFS* mit beliebiger Konnektivität. Für die niedrigste Detailstufe könnten wir eigentlich irgend ein Kompressionsverfahren verwenden. Wir haben uns aber für die *topological surgery* entschieden, weil die *PFS* Darstellung eine natürliche Erweiterung der *TS* Darstellung ist.

3.2 Das *Progressive Mesh* Verfahren (*PM*)

Dieses Verfahren kann als eine einfache Variante der *PFS* Kompriemierung angesehen werden. *PM* arbeitet mit bei beliebiger Konnektivität. Es ist ein adaptives Verfeinerungsschema, wo neue Flächen nicht durch Unterteilung existierender Flächen geschaffen werden, sondern indem sie zwischen die bestehenden Flächen eingefügt werden. Ein *vertex split* wird durch die Angabe von zwei Kanten mit einem gemeinsamen Eckpunkt spezifiziert. Das *mesh* wird nun so verfeinert, indem es an zwei Kanten entlang aufgeschnitten wird und der gemeinsame Knoten in zwei Knoten aufgeteilt wird. Dadurch entsteht eine quadratische Öffnung, die mit zwei Dreiecken so aufgefüllt wird, dass ihre gemeinsame Kante die zwei neuen Knoten verbindet.

3.3 Übersicht

Im Gegensatz zu *PM* wird in *PFS* die *vertex split*-Operation jeweils auf eine Menge von Knoten angewendet (vergleiche dazu Abbildungen 5 und 6). Diese Menge von Knoten gehören zu einem Baum von Kanten. Also anstatt das wir das Objekt an zwei Kanten aufschneiden, tun wir dies gerade entlang eines ganzen Baumes. Da wir mit einem Baum vielleicht nicht an genau denjenigen Stellen des Objektes eine Verfeinerung erzeugen können, wo dies erwünscht ist, müssen wir einen ganzen Wald von Bäumen angeben. In die entstandene Öffnung fügen wir die einfachen Polygone ein.

Eine *forest split* Operation wird durch einen Wald von Bäumen über den *mesh*punkten, eine Folge von einfachen Polygonen und eine Sequenz von Knotenverschiebungen definiert. Einige Informationen, um diese Schritte zu machen, werden nicht explizit gegeben, sondern beruhen auf impliziten Regeln zur Numerierung von Elementen.

3.4 Aufzählung von *mesh*elementen

Die Knoten werden von 0 bis $V-1$ und die Dreiecke von 0 bis $T-1$ numeriert. Die Kanten sind als Paare von Indizes (i,j) mit $i<j$ repräsentiert, in aufsteigender Reihenfolge sortiert und von 0 bis $E-1$ durchnumeriert. Diese Definitionen erlauben uns die Bäume des Waldes in aufsteigender Reihenfolge nach dem Blatt jedes Baumes mit kleinstem Index, dem *root vertex*, zu ordnen. Die *root edge* des Baumes ist diejenige Kante, die als einzige den *root vertex* als einen Endpunkt hat. Von den beiden Dreiecken, die an der *root edge* beteiligt sind, ist dasjenige Dreieck das *root triangle*, das den kleineren Index hat. Trennt man nun einen Baum den Kanten entlang auf, so entsteht eine Öffnung im Objekt. Der Rand der Öffnung kann in zyklischer Weise durchlaufen werden. Dieser Zyklus wird per Konvention beim *root vertex* beginnend in Richtung des *root triangle* traversiert. Beim Aufspalten des Baumes werden die Kanten verdoppelt, die *root edge* ist dann diejenige Kante, welche zum *root triangle* gehört. Ebenso wird für die Triangulierung des einzufügenden Polygons der *root vertex* und die *root edge* bestimmt.

Die Anzahl Kanten des einzufügenden Polygons muss natürlich derjenigen der Öffnung entsprechen; die Zuweisung der Knoten und Kanten, die zusammengebracht werden müssen, ist nun durch das Matching der *root vertex* und des Umlaufsinnes gegeben.

3.5 Die *forest split* Operation

Die *split* Operation auf einen Wald kann sequentiell durchgeführt werden, indem man einen Baum nach dem anderen bearbeitet. Die Kodierung der *forest split* Operation besteht aus der Kodierung der Kanten, der Sequenz von einfachen Polygonen und den Knotenverschiebungen.

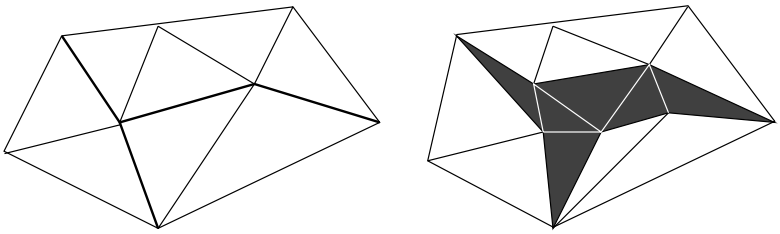

Abbildung 6: *Tree split* Operation

Die Zuordnung, welche Kanten zu einem Wald gehören kann sehr einfach gelöst werden. Mit jeder Kante wird ein Bit assoziiert, das bei zugehörigen Kanten den Wert 1 hat und 0 bei anderen. Diese Bits werden in der im Abschnitt 3.4 definierten Reihenfolge der Kantenindizes abgespeichert.

Nachdem wir die *cut edges* eines Baumes identifiziert haben, müssen wir die Indizes der bestehenden Dreiecke in die Knotenmenge aufdatieren, da beim *split* Knoten verdoppelt werden. Dazu beginnen wir beim *root triangle*. Wir können alle Ecken der betroffenen Dreiecke in der Reihenfolge des Randzyklusses besuchen, indem wir von Dreieck zu Dreieck springen und immer mit dem Baum in Kontakt bleiben. Dies generiert eine Liste von Dreiecksecken, genannt *corner loop*, deren Werte nachgeführt werden müssen. Wenn wir die Liste traversieren, erkennen wir Folgen von Ecken, die den selben Index hatten. Jeder dieser Folgen müssen wir genau einen neuen Index zuweisen. Um Lücken in der Knotennumerierung zu vermeiden, müssen wir alle alten Knotenindizes wiederverwenden. So erhalten wir den Schnitt in der Topologie.

Die entstandene Öffnung wird mit einem einfachen Polygon ausgefüllt. Die Umrandung der Öffnung hat natürlich genau gleich viele Knoten, wie das einfache Polygon. Jedes Dreieck der Triangulierung definiert ein Dreieck des verfeinerten *mesh*. Das wird gemacht, indem allen Indizes der Polygondreiecke die entsprechenden Indizes des Randes der Öffnung zugewiesen werden. Den neuen Knoten werden Koordinaten relativ zu den alten Knoten zugewiesen.

3.6 Kompression und Kodierung

In diesem Abschnitt beschreiben wir, wie ein Model im *PFS* Format kodiert ist. Für die niedrigste Auflösung haben wir dies im Kapitel 2 beschrieben. Dieser Datenblock wird gefolgt von der komprimierten Darstellung der *forest split* Operation in ihrer Reihenfolge, siehe Abschnitt 3.5.

Die Kodierung der einzufügenden einfachen Polygone kann auf zwei Arten gemacht werden. Das Verfahren aus Abschnitt 2.2 zusammen mit dem *marching pattern* wird *variable length coding* genannt und ist zur Beschreibung von Polygonen, die aus wenigen Dreiecken bestehen, oder für solche

mit kurzen *runs* ineffizient. Bei solchen Polygonen kann ein anderes Verfahren, genannt *constant length coding*, eingesetzt werden, das mit genau zwei Bits pro Dreieck auskommt und ebenfalls die gesamte Topologie bestimmt. Dazu wird der Baum der Polygondreiecke rekursiv durchlaufen. Für jedes Polygon wird der Wert der beiden Bits bestimmt. Wir beginnen mit den beiden Knoten der *root edge*. Der eine ist der *rechte Knoten* und der andere der *linke Knoten*. Mit den beiden verbleibenden Kanten des Dreiecks wird ein Bit assoziiert. Es wird auf 1 gesetzt, wenn die Kante eine innere Kante ist; im Falle einer Randkante wird es auf 0 gesetzt. Ist z.B. die linke Kante eine innere Kante, so wird der verbleibende Knoten des Dreiecks zum *rechten Knoten*, und umgekehrt. Sind beide Kanten innere Kanten, so legen wir den verbleibenden Knoten und den *rechten Knoten* auf einen Stapel und fahren mit den beiden anderen fort. Sind beide Kanten *boundary edges*, so haben wir ein Blatt erreicht, und wir nehmen die zwei obersten Knoten vom Stapel. Ist der Stapel leer, so sind wir fertig. Der Komprimierungsalgorithmus berechnet beide Verfahren und verwendet das effizientere.

Zu jedem einfachen Polygon müssen noch die Knotenverschiebungen kodiert werden.

3.7 Kompressionsrate

Die Effektivität der Verfahren können verglichen werden, indem man die durchschnittliche Anzahl Bits pro Dreieck, die für die Kodierung der *mesh*verfeinerung verwendet werden, bestimmt. Das *constant length coding* Verfahren dient uns als obere Grenze. Dieses benötigt für jedes neue Dreieck zwei Bits. Zusätzlich müssen wir noch die Kanten des Waldes markieren, dazu brauchen wir für alle Kannten ein Bit. Der beste Fall tritt dann ein, wenn der Wald aus einem spannenden Baum besteht. Er besitzt dann $V-1$ Kanten. Unter der Annahme der geschlossenen Objekte entspricht dies $2V-2$ *boundary edges* und somit $2V-4$ Dreiecken. Da typische *meshes* ungefähr doppelt so viele Dreiecke wie Knoten haben, entspricht dies ungefähr T. Weiterhin gilt mit Hilfe unserer Annahme, dass die Eulercharakteristik $V-E+T=0$ und somit $E=1.5T$ ist. Wir bezeichnen mit $\triangle T=\alpha T$ die Anzahl der bei dem *split* neu eingefügten Dreiecke. Damit folgt $E=1.5T=\frac{1.5\triangle T}{\alpha}$. Die Anzahl der Bits, die für die Konnektivitätsinformation eines *forest splits* verwendet werden, ist somit ungefähr gleich $(1.5/\alpha+2)\triangle T$. Das *PM* Verfahren benötigt vergleichsweise $(5+log_2(n))\triangle T$ Bits.

4 Generierung von verschiedenen Detailstufen

Das Modell steht uns nur in der höchsten Auflösung zur Verfügung. Wir benötigen das Modell aber in mehreren Auflösungen und zwar so, dass die einzelnen Stufen durch *forest splits* auseinander generiert werden können.

Wir zeigen, dass die meisten *edge collapse* basierten Algorithmen so modifiziert werden können, dass sie eine Folge von *forest collapse* Operationen bilden.

4.1 Die *forest collapse* Operation

Sie ist eigentlich die Umkehrung des *forest splits*. Die Menge der Dreiecke jeder Detailstufe, die in der nächst niedrigen zusammenfallen, können als verbundene Komponenten angesehen werden. Zwei Dreiecke sind verbunden, falls sie eine gemeinsame Kante besitzen. Eine Funktion, die definiert, in welcher Detailstufe welche Knoten zusammengelegt werden, heisst *clustering function*. Eine *clustering function* definiert eine *forest collapse* Operation, falls jede verbundene Komponente ein einfaches Polygon ist und kein Knoten an mehr als einer verbundenen Komponente beteiligt ist. Wenn diese beiden Bedingungen erfüllt sind, dann bilden die *boundary edges* dieser Komponenten in der nächst niedrigeren Detailstufe einen Wald von Bäumen.

4.2 *Edge collapse* Algorithmen

In einigen Algorithmen werden die Kanten des *mesh* nach einem bestimmten Kriterium, basierend auf einer Fehlernorm, in eine Prioritätswarteschlange eingeordnet. Die Fehlernorm gibt ein Mass für den Fehler an, der eingeführt wird, wenn man die Kante kollabiert. Die erste Kante wird dann aus der Warteschlange entfernt und einigen Tests unterworfen. Wenn sie die Tests besteht, wird die Kante entfernt und die beiden Endknoten zusammengelegt, sonst wird sie belassen. Das Zusammenlegen der Kante kann es nötig machen, die Reihenfolge in der Prioritätsschlange zu ändern. Dieser Prozess wird solange fortgesetzt bis die Warteschlange leer ist.

Das *mesh* zu Beginn definiert eine Detailstufe. Das Resultat eines vollen Durchlaufs des *edge collapse* Algorithmus ist dann die nächst niedrigere Detailstufe. Damit die Folge der *edge collapse* Operationen aber auch einen gültigen *forest collapse* ergeben, müssen bei jedem *edge collapse* noch zwei zusätzliche Tests vorgenommen werden. Beim Enfernen der Kanten verändern wir laufend die Konnektivität des *mesh*. Ob die Kriterien für einen *forest collapse* eingehalten werden, dazu müssen wir aber die Topologie des *mesh* beachten, wie sie vor den schon vorgenommenen *edge collapses* war.

Test 1 stellt sicher, dass mit dem Zusammenlegen der Knoten keine inneren Knoten entstehen.

Test 2 wird nur angewendet, wenn die Kante den ersten bestanden hat. Er verhindert, dass Knoten mehr als einer verbundenen Komponente angehört und ein Zyklus entstehen würde.

5 Implementation und Resultate

Unser Prototyp besteht aus drei Programmen: dem *Simplification*programm, dem Kodierungsprogramm und dem Dekodierungsprogramm. Die jetzige Implementation bearbeitet triangulierte *meshes* die eine Manigfaltikeit sind und bei welchen die Knoten keine weiteren Eigenschaften besitzen.

Das Verhältnis der Kompressionsraten von *PFS* und *single resolution* Verfahren liegt etwa bei 2 (zu Ungunsten von *PFS*). Pro Dreieck des Modells werden insgesamt etwa 20 Bits verwendet.

Zusammenfassend kann gesagt werden, dass *single resolution* Verfahren bessere Kompressionsraten besitzen als die *forest split* Operation. Die Komressionsraten nähern sich aber an, wenn die die Grösse der Bäume wächst und die Knotenverschiebungen kleiner werden.

Literatur

[1] A. Guéziec, G. Taubin, W. Horn, F. Lazarus. Progressive Forest Split Compression. *Siggraph '98 Conference Proceedings.*

[2] G. Taubin, J. Rossignac. Geometry Compression through Topological Surgery. *ACM Transactions on Graphics*, April 1998.