iPACKMAN
Texturkompression mit hoher Qualität und niedriger Komplexität für mobile Geräte

entwickelt von Jacob Ström und Tomas Akenine-Möller

Matthias Tkocz
Matrikelnummer: 20723808
scottsch@web.de
Master of Education
1. Semester

Computer Grafik
M.Inf.1261.Mp
WS 2013/14
Institut für Informatik
Georg-August-Universität Göttingen

Prof. Dr. Winfried Kurth

27. Februar 2014

Zusammenfassung

In dieser Arbeit stelle ich iPACKMAN, einen von Jacob Ström und Tomas Akenine-Möller entwickelten Kompressionsalgorithmus für Texturen vor. Es handelt sich dabei um eine Erweiterung des Vorgängers PACKMAN, die im Allgemeinen eine bessere Bildqualität liefert als aktuell verwendete Kompressionsalgorithmen mit niedriger Komplexität für die Dekompression.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Mobile technische Geräte leisten immer mehr auf sehr kleinem Raum, entsprechend werden auch an die Bildqualität hohe Ansprüche gestellt. Deswegen werden rasterbasierte Hardware-Architekturen für die Bildqualität verwendet, doch die Performance dieser Systeme wird in der Regel durch die verfügbare Bandbreite begrenzt [AMN03]. Aus diesem Grund werden Techniken entwickelt, um die Anforderung an die Bandbreite zu reduzieren. Eine davon ist die Texturkompression.

Bei dieser Technik werden Bilder komprimieren abgespeichert und beim Auslesen dekomprimiert, um das originale Bild wiederherzustellen. Diese Technik spart Bandbreite.

Ein gutes Texturkompressionssystem muss folgende Eigenschaften besitzen [SAM05]:

1. Die Anzahl der benötigten Gatter muss möglichst klein sein.

   Dies liegt an der Energieeffizienz, die für mobile Geräte wichtig ist. Niedrige Komplexität in der Dekomprimierung sorgt für wenig Gatter, was die Energieeffizienz steigert.

2. Eine fixierte Kompressionsrate wird benötigt.

   Da nicht alle Geräte identisch arbeiten, ist es notwendig eine allgemeine Basis zu haben. Zum Beispiel können Systeme an unterschiedlichen Punkten des Bildes mit der Dekomprimierung beginnen, darum ist eine einheitliche Kompressionsgröße notwendig.

3. Nachschlagetabellen müssen vermieden werden.

   Diese Eigenschaft schließt den Nutzen solcher Tabellen nicht aus. Es bedeutet, dass nicht jedes Bild seine eigene Nachschlagetabelle (auch „Look-up table“ genannt) haben darf. Diese Tabelle würde mitgesendet werden müssen und dadurch erneut Bandbreite verbrauchen.

4. Die Komprimierung sollte möglichst kurz sein.

   Dieser letzte Punkt hat keine hohe Priorität, weil dies als Vorarbeit von den Personen bereits erledigt wird, die das Bild zur Verfügung stellen wollen.

Ein solches Texturkompressionssystem ist PACKMAN. Jacob Ström und Tomas Akenine-Möller entwickelten dieses System und stellten dies 2004 in „SIGGRAPH 2004“ vor [SAM04]. Dieses System teilt ein Bild in Pixelblöcke

mit je 2 x 4 Pixeln auf. Die Pixel der Blöcke werden dann verschlüsselt, so dass jeder Block mit 32 Bits kodiert wird. Um dies mit anderen Systemen vergleichen zu können, misst man die benötigten Bits pro Pixel (BpP). In diesem Fall sind es 4 BpP.

PACKMAN hat bereits eine geringe Komplexität, aber die Bildqualität des Endproduktes konnte mit den heutigen Standards wie S3TC oder PVR-TC nicht mithalten, die ebenfalls 4 BpP verwenden. Entsprechend versuchten Stöm und Akenine-Möller eine verbesserte Version von PACKMAN zu entwickeln.

Das Ergebnis ist iPACKMAN und steht für improved (verbessertes) PACKMAN. Dabei wurden die Pixelblöcke von 2 x 4 auf 4 x 4 erweitert. Diese Änderung bietet iPACKMAN mehr Möglichkeiten, während weiterhin 4 BpP verwendet werden. iPACKMAN wurde 2005 in „SIGGRAPH 2005“ vorgestellt [SAM05]. Die in dieser Arbeit beschriebenen Informationen sind hauptsächlich aus diesem Bericht entnommen worden. Alle weiteren Quellen sind an entsprechender Stelle benannt.

Zunächst wird in Kapitel 2 die Funktionsweise vom originalen PACKMAN erläutert, um die grundlegende Funktionsweise näher zu bringen, da iPACKMAN darauf aufbaut. Im Anschluss daran wird iPACKMAN im 3. Kapitel detailliert erläutert, von der Grundidee über die Kompression bis hin zur Fehlermetrik. Das 4. Kapitel vergleicht dann iPACKMAN mit S3TC und PVR-TC und abschließend werden im 5. Kapitel Ideen für die Transparenz mit iPACKMAN vorgestellt.

2 PACKMAN

In diesem Kapitel wird die Funktionsweise der Texturkompressionstechnik PACKMAN im Detail erläutert [SAM04].

2.1 Die Grundidee

Die Grundidee besteht darin, das originale Bild mit Hilfe von zwei Komponenten wiederherzustellen. Im ersten Schritt wird das Bild in gleichgroße Blöcke von Pixeln geteilt und pro Block werden die beiden Komponenten erstellt. Die eine Komponente ist die Grundfarbe, die für einen ganzen Block gilt. Die zweite Komponente ist die „Leuchtdichte“. Dies ist ein Bitwert und ist für jeden Pixel unterschiedlich. Dieser Wert wird dem Bitwert der Grundfarbe hinzuaddiert. Dadurch entsteht eine neue Farbe für jeden einzelnen Pixel. Diese Endfarbe sollte möglichst nah an der Originalfarbe für den Pixel

Abbildung 2.1: Grundlegende Funktionsweise von PACKMAN [SAM05]

liegen. Abbildung 2.1 visualisiert diese Funktionsweise.

2.2 Funktionsweise

Im Folgenden wird nun das verfahren zu PACKMAN im Detail beschrieben.

PACKMAN unterteilt ein Bild in Blöcken mit 2 x 4 Pixeln und codiert diese Blöcke mit 32 Bits. Die Grundfarbe setzt sich aus den Farbtönen rot, grün und blau zusammen, die jeweils mit 8 Bits codiert sind. Bei der Kompression werden diese 8 Bitwerte auf 4 Bits reduziert. Somit benötigt ein Block 4 + 4 + 4 = 12 Bits um die Grundfarbe zu speichern. Um die Anzahl der Bits pro Farbe vereinfacht darzustellen wird RGBxyz als Notation eingeführt. Dabei steht x für die Anzahl der Bits der Farbe rot, y für grün und z für blau. Damit lautet die Standard 8 Bitcodierung RGB888 und die komprimierte 4 Bitversion RGB444.

Bei der Dekompression muss die Grundfarbe von RGB444 auf RGB888 erweitert werden. Dies erreicht man, indem die 4 Bitfolgen der einzelnen Farben verdoppelt werden. So wird aus $(1001)_2$ $(10011001)_2$. Damit können nicht alle Farben von RGB888 angesprochen werden, weil die komprimierte Version nicht stärker als mit 4 Bits differenzieren kann.

Nun sind 12 Bits von den 32 Bits belegt, wodurch noch 20 Bits für die Leuchtdichte zur Verfügung stehen. Um zu bestimmen, wie die Basisfarbe pro Pixel verändert werden muss, werden 16 vorgefertigte Tabellen verwendet. Diese sind universell und damit unabhängig von dem zu dekomprimierenden Bild. Für jeden 2 x 4 Block wird eine Tabelle mit Hilfe von 4 Bits ausgewählt.

Jede Tabelle besitzt 4 Einträge, wobei die ersten beiden Werte negativ sind und die letzten beiden Werte positiv. Außerdem ist der Betrag vom ersten und letzten Wert gleich, genauso wie der Betrag von dem zweiten und dritten Wert. Darüber hinaus ist der erste Wert im Betrag größer als der zweite Wert (vgl. Abbildung 2.2).

Diese 4 Tabelleneinträge stehen jedem Pixel separat zur Verfügung und

Tabelle	0	1	2	3	4	5	6	7
	-8	-12	-31	-34	-50	-47	-80	-127
	-2	-4	-6	-12	-8	-19	-28	-42
	2	4	6	12	8	19	28	42
	8	12	31	34	50	47	80	127

Abbildung 2.2: Die ersten 8 Tabellen von PACKMAN [SAM04]

jeder Pixel besitzt 2 Bits, um einen dieser Werte auszuwählen. Da in jedem 2 x 4 Block 8 Pixel sind, benötigen man insgesamt 16 Bits pro Block.

Der ausgewählte Tabellenwert wird dann auf die RGB888 Basisfarbe hinzuaddiert, um dann die Endfarbe für den Pixel im fertigen Bild zu erhalten. Somit ergeben sich dann mit der Grundfarbe (12 Bits), der Tabellenauswahl (4 Bits) und der Auswahl der Pixel (16 Bits) insgesamt 32 Bits.

Um die Funktionsweise der Dekompression etwas konkreter darzustellen, folgt nun ein Beispiel. Betrachtet wird der 32 Bit Binärcode von einem 2 x 4 Block eines Bildes:

$(0000\ 0010\ 1111\ 0000\ 00\ 01\ 10\ 10\ 00\ 01\ 01\ 00)_2$

Die ersten 12 Bits stehen für eine Grundfarbe, dementsprechend ist dann RGB444 = (0, 2, 15). Diese 4 Bit Zahlen werden nun durch Verdopplung der Folge auf 8 Bit erweitert. Dadurch ergibt sich RGB888 = (0, 34, 255). Dies ist nun die Basisfarbe für den gesamten Block. Die nächsten 4 Bits des Codes ergeben 0, womit dann die Tabelle 0 ausgewählt wird. Damit stehen den Pixeln im Block folgende Werte zur Verfügung: {-12, -4, 4, 12}

Im Binärcode gibt es nun 8 Paare aus 2 Bits, die pro Pixel einen Wert bestimmen. Der ersten Pixel oben links hat den Wert 0 somit wählt er den ersten Wert der Tabelle aus (-12). Dieser Wert wird nun zur Grundfarbe hinzuaddiert:

(0, 34, 255) + (-12, -12, -12) = (0, 22, 243)

Dabei ist zu beachten, dass der Wert einer einzelnen Farbe immer im Bereich zwischen 0 und 255 liegt. Sollte das Resultat die Grenze überschreiten, wird 0 oder 255 als Endwert gesetzt.

3 iPACKMAN

In diesem Kapitel wird die Texturkompressionstechnik von iPACKMAN im Detail erläutert. Zunächst wird die grundlegende Idee und die Motivation hinter der Technik in Kapitel 3.1 beschrieben. Danach wird in 3.2 die Dekompressionstechnik mit einem Hardwarebeispiel erklärt. 3.3 erläutert die möglichen Techniken zur Kompression. Im letzten Kapitel geht um die Fehlermetrik, die benötigt wird, um den Unterschied zwischen dem wiederhergestellten Bild und dem Original zu ermitteln.

3.1 Grunddesign und Motivation

Das größte Problem von PACKMAN wird in Bildern deutlich, die sehr viele fließende Farbübergänge, die so genannte Chrominanz, aufweisen. Selbst kleinste Sprünge der Chrominanz sind mit PACKMAN deutlich sichtbar zwischen den einzelnen Blöcken. Die Ursache dafür liegt in der RGB444 Codierung. Diese lässt mit 12 Bits nicht sehr viele Möglichkeiten für die Basisfarbe zu. Hinzu kommt, dass die 16 Tabellen für die Leuchtdichte nicht sehr viele Möglichkeiten bieten, die in einem so kleinen Wertebereichen nötig wären.

Das Ziel von iPACKMAN ist es, genau diese Chrominanzsprünge zu reduzieren. Um dies zu erreichen, experimentierte man mit den Blöcken. Zunächst fasste man die 2 x 4 Blöcke mit 32 Bits zu Paaren zusammen und man betrachtete dann die 4 x 4 Blöcke. Jeder Block besteht weiterhin aus zwei Hälften mit jeweils 32 Bits.

Die RGB444 codierten Hälften sind miteinander verglichen worden. Es stellte sich heraus, dass die Farben nicht immer sehr große Unterschiede aufzeigen. Um die Farbauswahl detaillierter zu gestalten, bekam jede Farbe ein weiteres Bit dazu. Damit besitzen die Farben nun eine RGB555 Codierung.

Mit der neuen Codierung wurden die Hälften miteinander verglichen. Dazu wurden die Differenzen der einzelnen Farben bestimmt mittels (R_1 – R_2 , G_1 – G_2 , B_1 – B_2). Das Ergebnis ist in Abbildung 3.1 sichtbar. Wie dieses Bild zeigt, ist die größte Spitze um den Wert 0. Dies bedeutet, dass sich die meisten Paare von 2 x 4 Blöcken nicht oder kaum unterscheiden. Wenn man eine Toleranz von [-4, 3] zulässt, sind 88% der Paare abgedeckt (und das ist mittels der schwarzen Balken sichtbar gemacht worden).

Der Grund für diese Toleranzgrenze liegt erneut in der Anzahl der Bits. Wenn man den ersten Block mit RGB555 codiert und der zweite Block in einem Intervall von [-4,3] von dem ersten Block abweicht, so kann man mit Hilfe von 3 Bits pro Farbe diesen Unterschied festhalten. Um die Farbe für die zweite Hälfte zu erhalten, wird auf die RGB555 Farbe der RGB333 Wert hinzu addiert. Damit hat man für jede Hälfte eine genauere Farbauswahl und

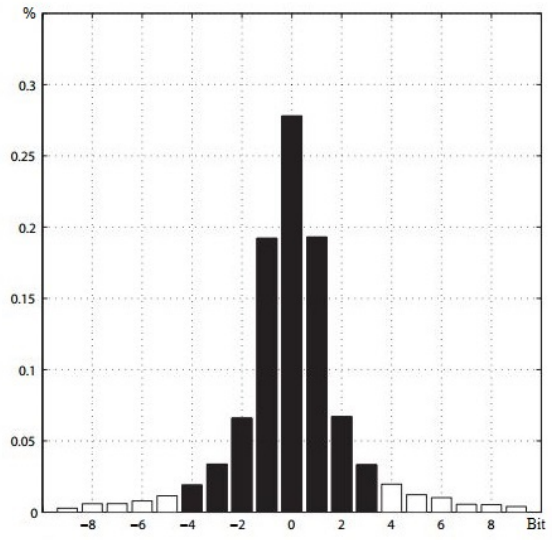


Abbildung 3.1: RGB555 Vergleichsresultate der Pixel [SAM05]

behält die ursprünglichen 24 Bits bei.

Dennoch lassen sich nicht alle Blöcke mit RGB555 codieren. 12% der Blockpaare hatten eine größere Differenz, als es im Intervall vorgesehen ist. Für diese Blöcke wurde beschlossen, weiterhin die RGB444 Codierung zu verwenden. Damit das System weiß, welche Codierung im 4 x 4 Block verwendet werden soll, muss ein Bit verwendet werden, welches diesen Umstand differenziert. Da die Farbauswahl 24 Bits benötigt und die einzelnen Pixel aus Tabellen mit vier Einträgen wählen sollten, wurde die Anzahl der Tabellen von 16 auf 8 reduziert. Diese Reduktion ließ die Bildqualität um 0,2 dB sinken, was überraschend niedrig ist, weil erst 0,25 dB einen sichtbaren Unterschied ausmachen.

Das freigewordene Bit wird nun als Differenzierungsbit (Diffbit) verwendet, um zwischen RGB555 und RGB444 umzuschalten. Des Weiteren besteht der 4 x 4 Block aus zwei Hälften und da jede Hälfte nun ein Bit weniger für die Tabellen hat, ist nach dem Diffbit ein weiteres Bit frei geworden. Dieses Bit wird ebenfalls für die Differenzierung verwendet. Es schaltet die Ausrichtung der Blöcke um. Die beiden Hälften können horizontal mit 4 x 2 zueinander stehen oder vertikal mit 2 x 4. Diese Umschaltung wird auch als „flippen“ bezeichnet und somit wird dieses Bit nun „Flipbit“ benannt.

Zusammengefasst würde das System von iPACKMAN wie folgt aussehen:

- Ein Diffbit schaltet zwischen RGB555 und RGB444 Codierung um.

- Ein Flipbit, um zwischen der horizontalen und vertikalen Anordnung.
- Zwei 3-Bit Codewörter für die Tabellenauswahl.
- 16 2-Bit Indices für die Wertauswahl für jeden einzelnen Pixel.
- 24 Bits für die Farbauswahl, entweder zwei mal mit RGB444 codiert oder einmal mit RGB555 + RGB333 codiert.

3.2 Dekompression

Um die Dekompression zu zeigen, ist in Abbildung 3.4 ein Hardwarebeispiel erstellt worden. Die Dekompression funktioniert im Detail folgendermaßen:

1. Im ersten Schritt wird die Grundfarbe bestimmt. Da das System für alle 3 Farben gleich ist, wird hier nur auf die rote Farbe im Detail eingegangen. Um die Farbe zu bestimmen, teilt das Diffbit dem System mit, dass nun RGB555 Codierung verwendet wird. Entsprechend wird R_5^1 verwendet und MUX (Multiplexer Einheit) U wählt für die erste Blockhälfte 0 oder für die zweite Blockhälfte R_3^2 aus. Dieser Wert wird dem anderen hinzu addiert. Bei RGB555 = (4, 15, 27) und RGB333 = (-4, -2, 3) wäre das Ergebnis (0, 13, 30) und würde für die zweite Blockhälfte verwendet werden.

   Wenn das Diffbit die RGB444 Codierung vorsieht, werden diese entsprechend ausgewählt. Da MUX V aber 5-Bit Werte erwartet, wird R_4^1 wie R_5^1 ausgewählt, wobei dann das fünfte Bit später ignoriert wird und für R_4^2 wird eine 0 hinzugefügt, die ebenfalls später ignoriert wird.

2. Der nächste Schritt ist die Erweiterung der Farbe auf 8 Bit. Der Erweiterer entscheidet mit Hilfe des Diffbits, ob RGB444 oder RGB555 Codierung vorliegt. Bei der 4 Bit Version wird die Folge einfach verdoppelt. So wird aus $(1011)_2$ $(1011\ 1011)_2$. Außerdem wird dabei das fünfte Bit weggeschnitten, welches durch MUX V mitgeliefert wurde.

   In der 5 Bit Version werden die letzten drei Bits kopiert und vorne eingefügt, zum Beispiel würde aus (0, 13, 30) (0, 107, 247) werden.

3. Um die korrekte Tabelle auszuwählen, schaltet MUX W das passende Tabellencodewort für die Blockhälfte durch. Dies besteht aus 3 Bits. Lautet der Wert $(011)_2$, so wird Tabelle 3 verwendet und den Pixeln stehen folgende Werte zur Verfügung: {-42, -13, 13, 42}.

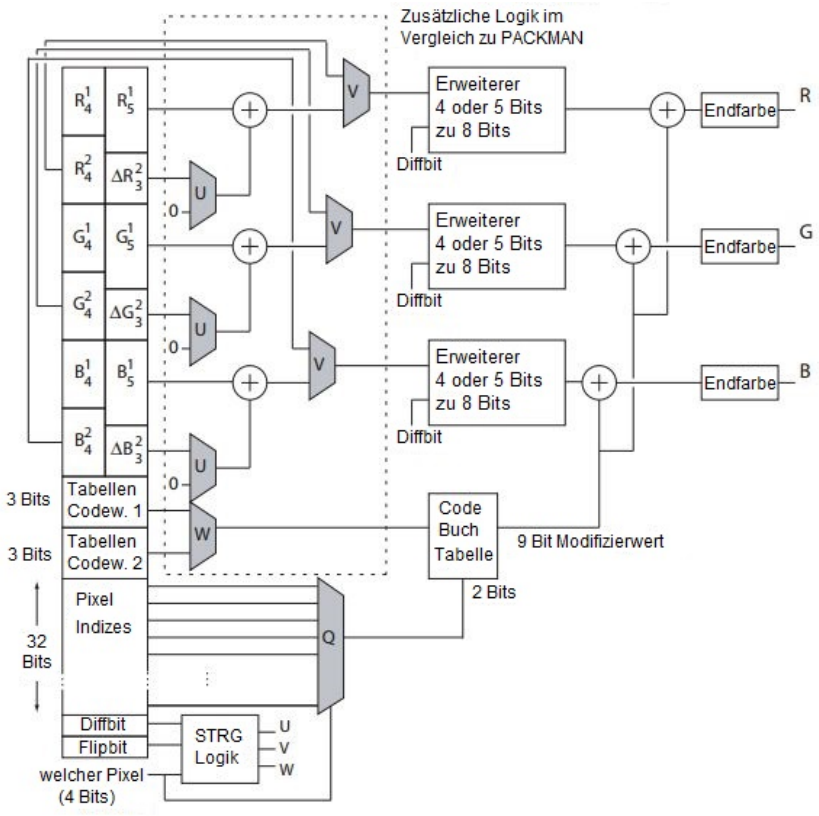


Abbildung 3.2: Mögliche Hardwareschaltung für iPACKMAN [SAM05]

4. MUX Q schaltet die Indices eines der 32 Pixel durch. So würde zum Beispiel der Pixel A den Wert 3 mitbringen. Dadurch würde aus der Tabelle der Wert 42 ausgewählt werden.

5. Im letzten Schritt werden die Werte zusammenaddiert, um für den Pixel A die Endfarbe zu ermitteln. Wichtig dabei ist, dass die Werte im Bereich [0, 255] liegen müssen und bei Überschreitung der grenzen 0 oder 255 gesetzt wird.

   Die oberen Beispiele würden Folgendes ergeben: (0, 107, 247) + (42, 42, 42) = (42, 149, 255)

Zur Vervollständigung des Hardwarebeispiels ist der Steuerwerk W zu benennen. W besteht aus $w_3w_2w_1w_1$ und gibt an, welcher Pixel aktuell dekomprimiert wird. Entsprechend wird dieser Wert in die Steuereinheit hinein geleitet und schaltet die MUX Einheiten um.

Die verwendeten Werte in den Tabellen wurden unter Verwendung von Trainingsbildern generiert. Dazu hat man mit zufälligen Werten begonnen

und dann diese Werte optimiert, so dass unter den Trainingsbildern die Fehler minimal wurden.

3.3 Kompression

Im originalen PACKMAN hat man 2^{12} verschiedene Farben, 2^4 Tabellen und 2^2 Möglichkeiten für einen Pixel. Da dieser Bereich noch relativ klein ist, durchsucht ein Algorithmus alle Möglichkeiten und wählt am Ende die Einstellungen aus, die das beste Ergebnis liefern.

iPACKMAN hat für die Farben 2^{24} verschiedene Möglichkeiten. Zusätzlich besteht jeder Block aus zwei verschiedene Hälften, die sich aufeinander beziehen. Durch die weit größere Auswahl ist die komplette Durchsuchung der Möglichkeiten weniger sinnvoll und sehr zeitraubend. Aus diesem Grund wurden drei Techniken entwickelt, um Bilder zu komprimieren, die nun im einzelnen vorgestellt werden. Alle Techniken hatten das gleiche 128 x 128 Bild als Vorgabe und ein 1,2 Ghz Laptop wurde dafür verwendet.

3.3.1 Schnellste Möglichkeit

Nachdem alle Blöcke eingeteilt sind, werden alle Unterblöcke mit RGB555 codiert und die Differenz innerhalb der einzelnen Blöcke ermittelt. Liegt diese Differenz im Intervall [-4, 3], so wird die Codierung beibehalten und übernommen. Für die Leuchtdichte werden dann die Tabellen auf diese Blöcke angewendet. Es werden alle Möglichkeiten durchsucht und am Ende wird die Tabelle ausgewählt mit den geringsten Fehlern zum Originalbild. Anschließend wird noch das Flipbit einbezogen und auch dafür alles getestet. Das beste Ergebnis wird am Ende genommen

Für die Blöcke mit einer größeren Differenz als im Intervall vorgesehen, wird die RGB444 Codierung verwendet. Dabei durchsucht ein Algorithmus alle Tabellen und das Flipbit nach der besten Lösung.

Diese Technik benötigte 60 Millisekunden.

3.3.2 Mittelweg zwischen Qualität und Geschwindigkeit

Bei dieser Methode wird die gleiche Methode wie bei der schnellsten verwendet mit einigen kleineren Änderungen. Bei den RGB555 Blöcken wird eine Farbtoleranz von ± 1 bei der Durchsuchung erlaubt. Das bedeutet, dass der Durchsuchungsraum minimal größer wird. Bei den RGB444 codierten Blöcken wird die alte Durchsuchungstechnik wie beim originalen PACKMAN verwendet.

Diese Technik benötigte 20 Sekunden für das Bild.

3.3.3 Qualitative beste Möglichkeit

Der Unterschied in der Farbe der RGB444 codierten Blöcke ist nicht immer sehr groß. Wenn man diese Blöcke etwas näher zusammenrückt, kann RGB555 Codierung möglich werden. Aus diesem Grund wird für diese Blöcke ein Intervall von [-9, 8] gesetzt. Sind Werte innerhalb dieses Intervalls wird versucht eine passende RGB555 Codierung zu finden. Zusätzlich wird eine RGB444 Lösung erstellt. Am Ende gewinnt das beste Resultat zwischen den beiden Möglichkeiten für den einzelnen Block. Ansonsten funktioniert die Technik wie die zweite.

Diese Technik ist sehr langsam und benötigte etwa sieben Minuten für das 128 x 128 Bild.

3.3.4 Ergebnis

Eine schnelle Kompression ist in allen Auflösungen schnell durchgeführt, aber es kann bei großen Farbübergängen ungenau werden. Die qualitativ hochwertigste Methode dauert zu lange, um sie in der Praxis zu verwenden. Der Mittelwert zwischen Geschwindigkeit und Qualität arbeitet langsam, aber in einer guten Zeit, wenn mehr Wert auf etwas bessere Qualität gelegt wird.

3.4 Fehlermetrik

Bei der Kompression wurde davon gesprochen, dass „das beste Resultat“ oder „mit minimalen Fehlern“ ausgewählt wurde. Damit das System weiß, welches dieses „beste Resultat“ ist, gibt es eine Fehlermetrik. Diese Metrik vergleicht das dekomprimierte Bild mit dem Original und misst den Unterschied zwischen den Farben rot, grün und blau. Dadurch kann man Werte vorsehen, die in der Toleranz liegen für die Kompression, um das beste Resultat zu erzielen.

Prinzipiell muss man für die Leuchtdichte eine Skalierung k finden, so dass die Farbe **b** addiert mit der Modifikation k(1,1,1) so nah wie möglich an der geforderten Farbe **d** ist. Als Formel ausgedrückt:

$$\mathbf{b} + \mathrm{k}(1\ ,\ 1\ ,\ 1) \approx \mathbf{d}$$

Um dies genauer zu erläutern, gibt es zwei Farben $u = (u_r, u_g, z_b)$ und $v = (v_r, v_g, v_b)$. Eine simple Fehlermetrik würde wie folgt aussehen:

$$e^2_{normal}(u, v) = (u_r - v_r)^2 + (u_g - v_g)^2 + (u_b - v_b)^2$$

Das optimale k wird dann gefunden, indem man die Differenz von d – b bildet und diese nach (1 , 1 , 1) projiziert.

Da das menschliche Auge empfindlicher auf Grün reagiert, als auf Rot oder Blau, muss die simple Fehlermetrik gewichtet werden. Dies sieht wie folgt aus:

$$e^2_{wahrnehm}(u,v) = w_r^2(u_r - v_r)^2 + w_g^2(u_g - v_g)^2 + w_b^2(u_b - v_b)^2$$

wobei $w_r^2 + w_g^2 + w_b^2 = 1$ und w_g größer sein kann als w_r und w_b.

Gewählt wurde die typische Gewichtung, wie im Farbfernsehen. Diese Auswahl erzielte den gewünschten Effekt bei der Leuchtdichte wie das Originalbild. Die Werte dafür lauten:

$w_r^2 = 0,299$, $w_g^2 = 0,587$, $w_b^2 = 0,114$

4 Vergleiche

Dieses Kapitel vergleicht verschiedene Texturkompressionstechniken miteinander. Das neue iPACKMAN wird mit PACKMAN, S3TC und die 4-Bit Version von PVR-TC verglichen. Zunächst werden die anderen Texturkompressionstechniken grob vorgestellt und anschließend die Resultate der Vergleiche vorgestellt.

4.1 S3TC

S3TC steht für 'S3 Texture Compression' und ist von Zhou Hong, Konstantine I. Iourcha und Krishna S. Nayak erfunden und patentiert worden. Dieses Texturkompressionssystem ist in DirectX 6 aufgenommen worden und wird noch heute als Standard verwendet. In Direct3D 10 ist dieses System als veraltet deklariert worden.

S3TC existiert in fünf verschiedenen Formaten (DXT1 bis DXT5). Der Unterschied der fünf Formate liegt in der Handhabung der Transparenz der Texturen. Die Basisfarbgebung ist bei allen Formaten gleich und funktioniert grundlegend wie folgt:

Das Bild wird in 4 x 4 Pixelblöcke gerastert, so dass 16 Pixel pro Block betrachtet werden und eine Speicherung von 4BpP benötigt wird. Aus 16 Farbwerten werden zwei 16 Bit codierte Farben berechnet, so dass es eine RGB565 Codierung gibt. Diese beiden Farben werden mittels Algorithmen in weitere Farben zerlegt und in einer temporären Tabelle gespeichert. Abschließend wird mit wenigen Bits aus dieser Tabelle für jeden Pixel eine Farbe ausgewählt [INH99].

4.2 PVR-TC

PVR-TC ist ein Texturkompressionssystem, das von Simon Fenney entwickelt wurde. Es existieren zwei Formate zu diesem System, eine 2 BpP Version und eine 4 BpP Version. Für den Vergleich mit den anderen Systemen ist man nur auf die 4 BpP Version eingegangen, da die anderen Systeme ebenfalls damit operieren.

Bei PVR-TC wird das Bild in zwei kleinere Bilder gespeichert, mit einer kleineren Skalierung und hat eine schlechtere Bildqualität. Beim Dekomprimieren werden die Bilder bilinear hochskaliert und anschließend übereinander gelegt. Mit kurzen Berechnungen entsteht dann das vollständige Bild in hoher Qualität.

Dieses System wird aktuell in allen mobilen Mac-Produkten für die Grafikausgabe verwendet. Da von diesem System kein öffentlich zugänglicher Codec existiert, konnte dieses System nur theoretisch verglichen werden [Fen03].

4.3 Resultate

In diesem Abschnitt werden nun die Texturkompressionssysteme PACKMAN, iPACKMAN und S3TC miteinander verglichen. Als Vergleichsbasis wurden 7 Bilder verwendet. Da diese nicht quadratisch waren, wurde jeweils die obere linke Ecke gewählt mit 512 x 512 Pixeln. Keines der Bilder war zuvor eine Entwicklungsumgebung eines der Systeme. Für die Testumgebung wurden die ersten fünf Bilder von einer Beispielseite von Kodak [Kod14] verwendet und zwei weitere, namentlich Lena [Law14] und Lorikeet [fli14]. Diese Bilder sind auch als Sammlung im Anhang zu finden.

	Kodak img 1	Kodak img 2	Kodak img 3	Kodak img 4	Kodak img 5	Lena	Lorikeet	Avg gain
PACKMAN	33.81	34.00	35.37	35.50	32.35	33.56	31.73	+2.54 dB
S3TC	34.78	36.82	38.53	37.96	32.80	35.97	34.37	+0.41 dB
PVR-TC	33.8 [8.98]	37.1 [6.20]	37.9 [5.61]	37.7 [5.76]	32.4 [10.59]	35.9 [7.11]	34.8 [8.08]	+0.65 dB
*i*PACKMAN	36.29	38.08	38.62	38.59	34.12	35.17	33.25	—

Abbildung 4.1: Tabellarische Resultate des Vergleichs [SAM05]

Die Resultate der Vergleiche sind tabellarisch in Abbildung 4.1 und grafisch in Abbildung 4.2 zu sehen. Man sieht, dass PACKMAN in allen Bildern deutlich schlechter abgeschnitten hat, als die anderen Techniken. Dieser Vergleich dient mehr zur Verdeutlichung, wie sehr sich iPACKMAN verbessert hat. In fünf der sieben Bildern lieferte iPACKMAN sichtbar bessere Qualität als die anderen beiden Techniken.

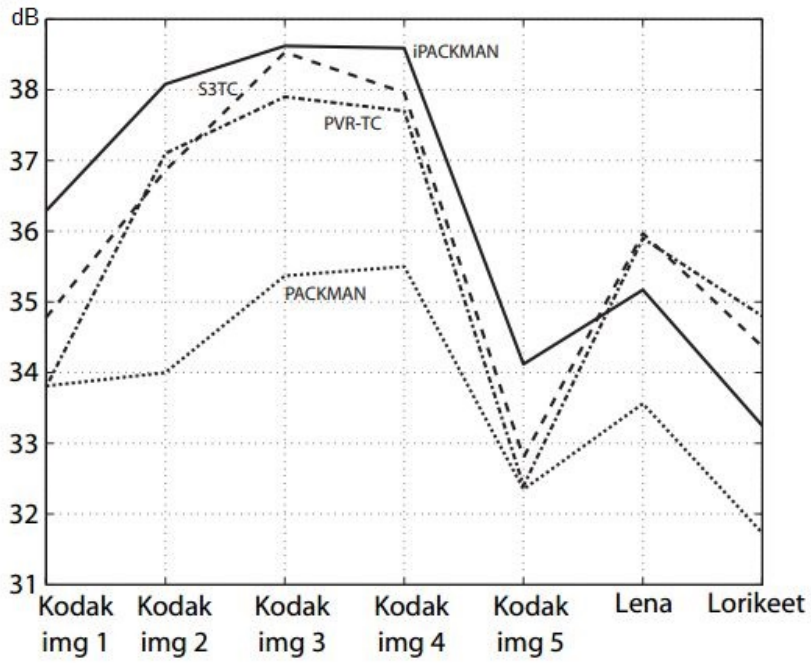


Abbildung 4.2: Grafische Resultate des Vergleichs [SAM05]

Dennoch hat jedes System seine individuellen Stärken und Schwächen. Abbildung 4.3 zeigt mehrere Vergleichsbilder zwischen den Techniken. Keines der Bilder war zuvor eine Testumgebung. Sichtbar schlechter war iPACKMAN vor allem bei Lorikeet in der ersten Reihe. In diesem Bild gibt es sehr viele fließende Chrominanzen und sieht man deutlich die Schwäche von iPACKMAN in diesem Bereich. S3TC kann diese Farbübergänge genauer abbilden und erreicht damit eine bessere Bildqualität.

Reihe 2 zeigt eine Textur für Computerspiele und dabei schneiden alle Systeme ähnlich gut ab. Die dritte Reihe stellt die Stärken von iPACKMAN heraus. Da die Basisfarbe relativ gleich bleibt, kann iPACKMAN mit seiner Leuchtdichte, die nötigen Korrekturen vornehmen, um mehr Farbunterschiede machen zu können im Bereich der einzelnen Pixel. Dies wird besonders um die Augenpartie herum deutlich. S3TC bildet in den Bereichen größere Pixelblöcke ab, was auf dem Monitor deutlich sichtbarer sein kann als in gedruckter Form.

Zusammengefasst ergab die Testreihe, dass iPACKMAN im Durchschnitt um 2,54 dB besser war als das alte PACKMAN, um 0,65 dB besser als PVR-TC und um 0,41 dB besser als S3TC. Da bereits 0,25 dB einen sichtbaren Unterschied ausmachen, sind diese Werte nicht unerheblich. Dennoch hat

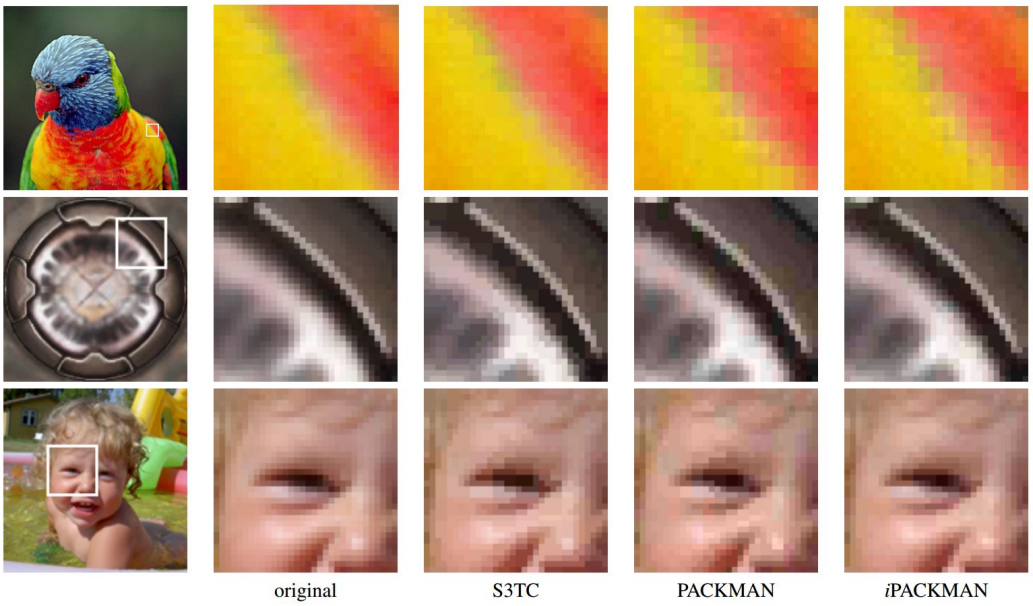


Abbildung 4.3: Bilder im Detailvergleich. [SAM05]

iPACKMAN weiterhin seine Schwächen, vor allem im Bereich der fließenden Chrominanz, wie Lorikeet gezeigt hat.

5 Transparenz

Texturkompressionstechniken haben in der Regel eine Möglichkeit, um mit Transparenz umzugehen. Entsprechend mehr Informationen benötigen diese Bilder. Auch iPACKMAN hat zwei Ideen, um dies umzusetzen.

Der Alphawert bestimmt, wie transparent ein Pixel sein kann. Die simpelste Methode ist es, für jeden Pixel 4 Bits zu verwenden, die einen Alphawert angeben. Dies bedeutet, dass 4 x 4 x 4 = 64 Bits pro Block verwendet werden müssen. Daraus folgt, dass jeder Pixel nur 16 Transparenzstufen kennt. Diese Technik würde die 4 BpP auf 8 BpP verdoppeln. DXT2 und DXT3 sind Transparenzformate von S3TC und sie verwenden diese Methode.

Eine andere Möglichkeit für iPACKMAN besteht darin, auch für die Transparenz eine ähnliche Vorgehensweise zu verwenden wie bei der Farbauswahl. Anstelle einer RGB555 oder RGB444 codierten Farbe verwendet man nun zwei Basisalphawerte pro Block. Entweder hat man hier 5 + 3 Bits oder zwei Mal die 4 Bits, die dann mit einem Diffbit unterschieden werden. Außerdem

gibt es pro Hälfte ein 3 Bit Tabellencodewort. Da die Basisalphawerte weniger Informationen benötigen, stehen den einzelnen Pixeln je ein Bit mehr zur Verfügung (insgesamt 48 Bits). Dadurch besitzen die Tabellen nun doppelt so viele Einträge. Dies würde ebenfalls in einer 8 BpP Gewichtung resultieren und hätte mehr Abstufungen als der simple Vorschlag.

Hinzu kommt, dass diese zweite Technik näher an DXT4 und DXT5 von S3TC liegt, weil auch dort die Transparenzcodierung der Farbcodierung ähnelt.

Anzumerken ist, dass dies noch in der Forschung ist und somit die Entwicklung noch nicht abgeschlossen ist.

6 Fazit und Ausblick

Für mobile Geräte wie Spielekonsolen oder Smartphones sind wenig Bandbreite, niedriger Energiebedarf sowie möglichst wenig Platzverbrauch wichtig. iPACKMAN erfüllt diese Bedingungen und hat gezeigt, dass es im Durchschnitt stärker ist als die aktuell verwendeten Methoden. Der große Vorteil ist ebenfalls, dass diese Technik sich nicht nur auf mobile Geräte beschränkt. Es könnte ebenfalls für Desktopgrafiken verwendet werden.

Trotz der ganzen Ergebnisse lässt sich iPACKMAN noch immer verbessern. Aktuell werden für RGB555 und RGB444 die gleichen Tabellen verwendet, um die Leuchtdichte zu bestimmen. Da die Farbübergänge zwischen den RGB555 Blöcken nicht sonderlich groß sind, benötigen sie eher kleinere Werte in den Tabellen. Die RGB444 codierten Blöcke benötigen dagegen eher große Werte.

Um iPACKMAN diesbezüglich weiter zu verbessern, kann man die Tabellen durch das Diffbit zusätzlich unterscheiden lassen. So würde man für RGB555 kleinere Werte auswählen können und für RGB444 größere. Außerdem muss noch die Technik für die Transparenz vervollständigt werden.

Literatur

[AMN03] T. Aila, V. Miettinen, and P. Nordlund. Delay streams for grapics hardware. *ACM Transactions on Graphics*, 22:792–800, 2003.

[Fen03] S. Fenney. Texture compression using low-frequency signal modulation. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (HWWS '03)*, pages 84–91, 2003.

[fli14] flickr. Lorikeet-512. *http://www.flickr.com/photos/60488854@N07/5518629434/*, Letztes Mal aufgerufen: Februar 2014.

[INH99] K. Iourcha, K. Nayak, and Z. Hong. System and method for fixed-rate block-based image compression with inferred pixel values. *US Patent 5956431 A*, 09 1999.

[Kod14] Kodak. Kodak lossless true color image suite. *http://r0k.us/graphics/kodak/*, Letztes Mal aufgerufen: Februar 2014.

[Law14] G. R. Lawrence. Lena. *http://en.wikipedia.org/wiki/Lenna*, Letztes Manlaufgerufen: Februar 2014.

[SAM04] Jacob Ström and Tomas Akenine-Möller. Packman: texture compression for mobile phones. *SIGGRAPH '04 ACM SIGGRAPH 2004 Sketches*, page 66, 2004.

[SAM05] Jacob Ström and Tomas Akenine-Möller. ipackman: High-quality, low-complexity texture compression for mobile phones. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware (HWWS '05)*, pages 63–70, 2005.

A Anhang

Hier befindet sich die Sammlung der Bilder, die für den Vergleichstest verwendet werden.

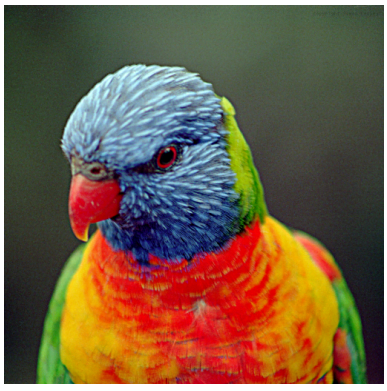
Kodak img 1 [Kod14]

Kodak img 2 [Kod14]

Kodak img 3 [Kod14]

Kodak img 4 [Kod14]

Kodak img 5 [Kod14]



Lorikeet [fli14]

Lena [Law14]