iPACKMAN
High-Quality, Low-Complexity Texture Compression for Mobile Phones

von Jacob Ström und Tomas Akenine-Möller

aus
„Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware (HWWS '05)“

Präsentiert von Matthias Tkocz

Einleitung (1/2)

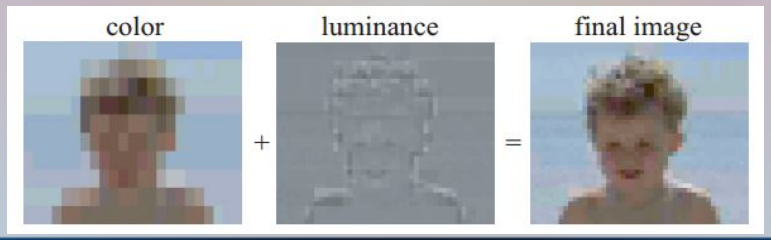
- Mobile Geräte können immer mehr auf kleinerem Raum
- Raster-basierte Hardware-Architekturen werden dafür verwendet
- Diese sollen qualitativ hochwertige Bilder liefern können
- Problem: Die verfügbare Bandbreite limitiert die Performance dieser Hardware
- Grundidee der Texturkompression:
  - Die Bilder als komprimierte Versionen der Textur zu speichern
  - Beim Rendern laufen sie über den Bus und werden gleichzeitig dekomprimiert
  - Dies spart Bandbreite

Einleitung (2/2)

- Folgende Eigenschaften benötigt dieses System:
  - Die benötigten Gatter sollten minimal sein (Effizienz)
  - Fixierte Kompressionsrate (Bits pro Pixel) wird benötigt (Allgemeinheit)
  - Spezielle Look-up Tabellen sollten vermieden werden (zusätzliche Daten)
  - Komprimierung sollte möglichst kurz sein (Vorarbeit)

PACKMAN

- Ein älteres Texturkompressions-System und dient als Basis für iPACKMAN
- Funktionsweise:
  - Das Bild wird in 2 x 4 Blöcke eingeteilt und durch 32 Bits kodiert
  - 12 Bits für die Grundfarbe → RGB444
  - 20 Bits für die Leuchtdichte bzw. Farbmodellierung
  - Es gibt Wertetabellen, die den Farbwert bearbeiten
  - 4 Bits bestimmen die Tabelle (16 Stück)
  - Jede Tabelle hat 4 Einträge
  - Jeder Pixel im Block hat seinen eigenen Wert (2 Bits pro Pixel)
  - In einem Block sind 8 Pixel → 16 Bits für die Farbanpassung



PACKMAN - Dekompressionsbeispiel

- Konkretes Beispiel für einen einzelnen Pixel
- 12 Bits bestimmen die Grundfarbe: RGB = (0 , 2 , 15)
- Dieser Wert wird von 4 Bits auf 8 Bits erweitert: RGB = (0 , 34 , 255)
- Mit den 4 Bit-Tabellencodewort wird die Tabelle ausgewählt:
  Tabelle 1: {-12 , -4 , 4 , 12}
- Jeder Pixel hat 2 Bits um einen dieser Werte auszuwählen: Bei 0 → -12
- Dieser Wert wird nun zu der Grundfarbe addiert:

  (0 , 34 , 255) + (-12 , -12 , -12) = (0 , 22 , 243)
- Zu beachten: Die Werte bleiben zwischen 0 und 255

table codeword	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

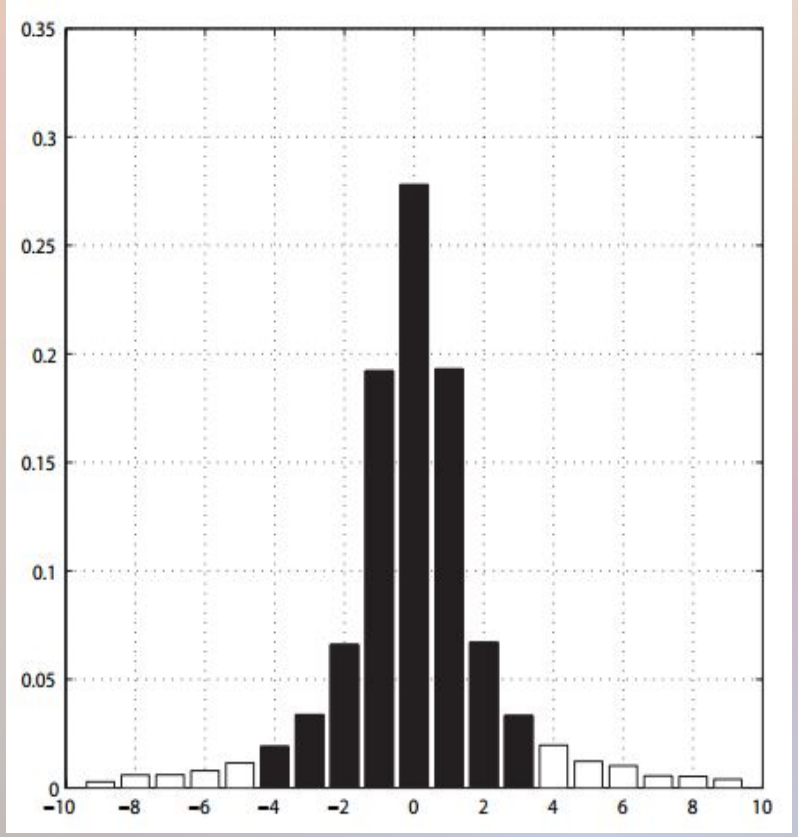
iPACKMAN – Grundidee (1/2)

- Ziel war es PACKMAN zu verbessern
- Das Problem waren die 12 Bits für die Basisfarbe
- Kleinste Sprünge in der Chrominanz waren deutlich sichtbar zwischen den Blöcken

- Diese Sprünge sollten reduziert werden
- Dazu wurden die Blöcke von 2 x 4 auf 4 x 4 vergrößert
- Die Farbe wurde in Relation dazu gesetzt
- Die Anzahl der Bits wurde ebenfalls erhöht, für genauere Farbauswahl
- RGB444 → RGB555

iPACKMAN – Grundidee (2/2)

- Paare von 2 x 4 Blöcken wurden auf RGB555 kodiert
- Deren Unterschied wurde wie folgt ermittelt: ($R_1 – R_2$, $G_1 – G_2$, $B_1 – B_2$)
- Ergebnis: Viele Blöcke unterscheiden sich kaum voneinander
- Wenn man eine Toleranz von -4 bis 3 zulässt passen 88% aller Blockpaare darauf
- Darum wurde RGB555 für die erste Hälfte gewählt und die 3 Bits als Modifikation für die zweite Hälfte.
- 12% der Blockpaare passen nicht in dieses Schema und es wurde beschlossen diese weiterhin die alte Weise mit RGB444 zu verwenden

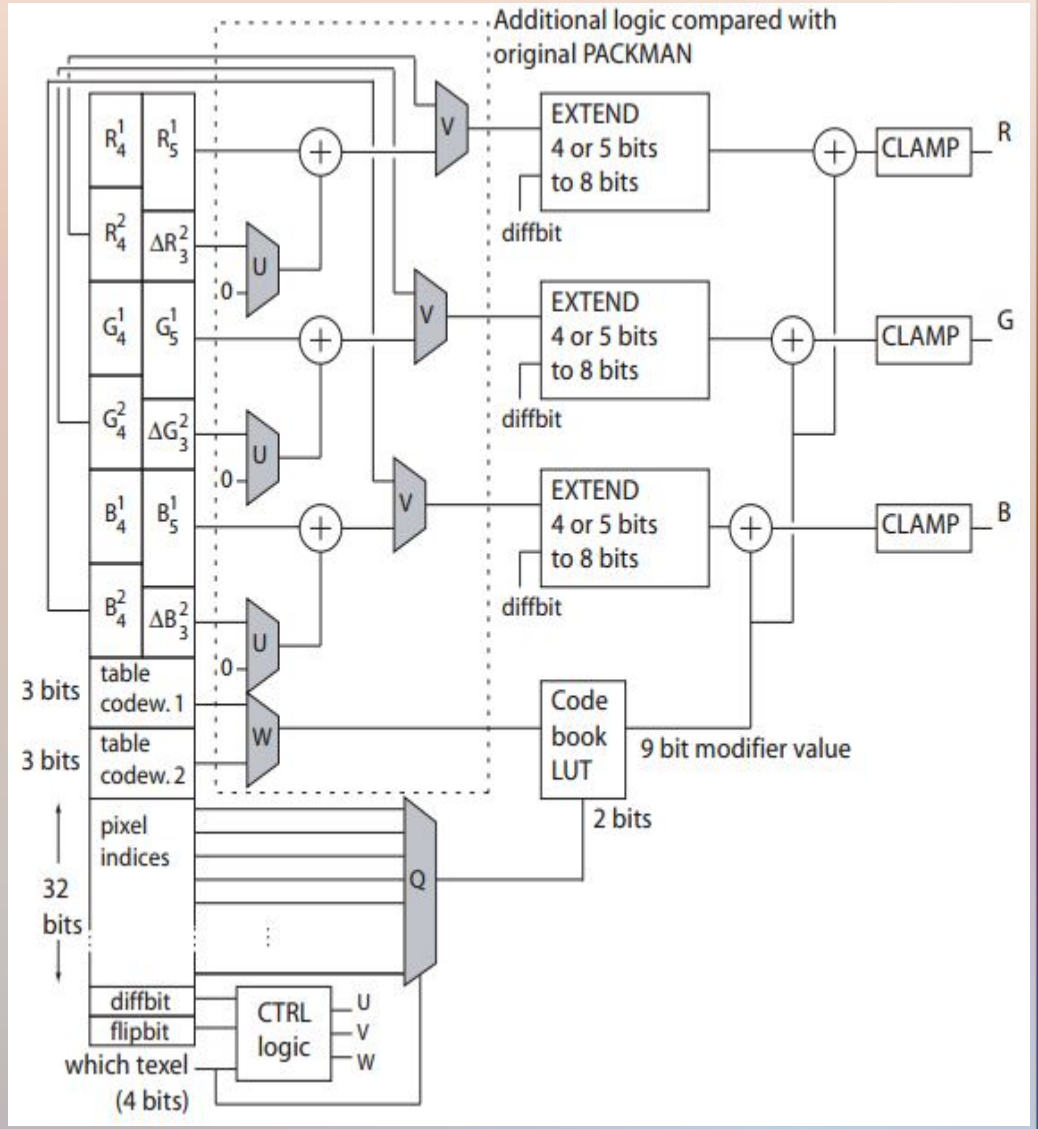


iPACKMAN - Funktionsweise

- Das Bild wird in 4 x 4 Blöcke eingeteilt und durch 64 Bits kodiert
- 24 Bits für die Grundfarbe → RGB555 + 3 Bits oder 2x RGB444
- 1 Diffbit wurde eingefügt zur Differenzierung der RGBs
- Das Diffbit musste irgendwo herkommen und es wurden die Tabellen ausgewählt
- 2 Mal 3 Bits bestimmen die Tabelle pro Blockhälfte (8 Stück)
- Jede Tabelle hat 4 Einträge
- Jeder Pixel im Block hat seinen eigenen Wert (2 Bits pro Pixel)
- In einem Block sind 16 Pixel → 32 Bits für die Farbanpassung
- 63 Bits sind belegt → letzte Bit kann für andere Dinge verwendet werden
- 1 Flipbit: Es entscheidet, ob die Hälften horizontal oder vertikal angeordnet sind

iPACKMAN – Hardwarebeispiel

- MUX U steuert RGB555
- Farbwahl durch MUX V (RGB444 wird als 5 Bits behandelt)
- RGB wird auf 8 Bits erweitert
- Diffbit bestimmt ob 4 oder 5 Bit
- MUX W bestimmt die Tabelle
- MUX Q steuert die Wahl der Farbmodifikation
- Am Ende wird alles addiert
- w bestimmt den Texel
- Beispiel: RGB555 = (4 , 15 , 27) und RGB333 = (-4 , -2 , 3)
  mit dem Tabellenwert 42
  → (0 , 13 , 30) → (0 , 107 , 247)
  → (42 , 149 , 255)



iPACKMAN – Kompression (1/2)

- PACKMAN hatte 2^12 Farben 2^4 Tabellen und 2^2 Werte
- Es wurde alles getestet was das beste Ergebnis lieferte
- iPACKMAN besitzt 2^24 Farben, was das ganze ineffizient macht
- 3 Schemen wurden ausprobiert bei einem 128 x 128 Bild mit einem 1,2 GHz Laptop
- Schnellste mit 60 Millisekunden:
  - Alle differenzierbaren Blöcke (RGB555) wurden gewählt und im Verhältnis zueinander kodiert
  - Alle Tabelleneinträge durchsucht für das beste Ergebnis (auch das Flipbit)
  - Alle nicht differenzierbaren Blöcke wurden als RGB444 gespeichert
    - Hier wurde auch nur das grobe Verhältnis gewählt

iPACKMAN – Kompression (2/2)

- Mittelweg zwischen Qualität und Geschwindigkeit mit 20 Sekunden:
  - Wie die schnellste Methode
  - Unterschied bei der Kodierung der nicht differenzierbaren Blöcke:
    - Bei RGB444 wurde nicht das Verhältnis gewählt, sondern jede Möglichkeit wurde getestet
- Qualität mit etwa 7 Minuten:
  - Für die RGB555 Kodierung erweitert man die Differenz
  - Man sucht Farbdifferenzen im Bereich [-9 , 8]
  - Nicht differenzierbare Blöcke wurden erneut durchlaufen.

iPACKMAN – Fehlermetrik (1/2)

- Die Fehlermetrik bestimmt die Toleranz und somit die Leuchtdichte der Farben
- Es wird ein Skalar k gesucht so dass eine Farbe B mit der Modifikation k(1 , 1 , 1) so nah wie möglich an der geforderten Farbe D ist:

$$B + k(1, 1, 1) \sim D$$

- Für 2 Farben U und V könnte es 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 gefunden durch:

$$D - B = (1 , 1 , 1)$$

iPACKMAN – Fehlermetrik (2/2)

- Da das menschliche Auge empfindlicher auf grün reagiert als auf rot oder blau wird die Formel gewichtet:

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

- Zu beachten: $w_r^2 + w_g^2 + w_b^2 = 1$
- Gewählt wurde die typische Gewichtung wie im Farbfernsehen, da diese den gewünschten Effekt erzielte und die gleiche Leuchtdichte erzielte wie das Original:

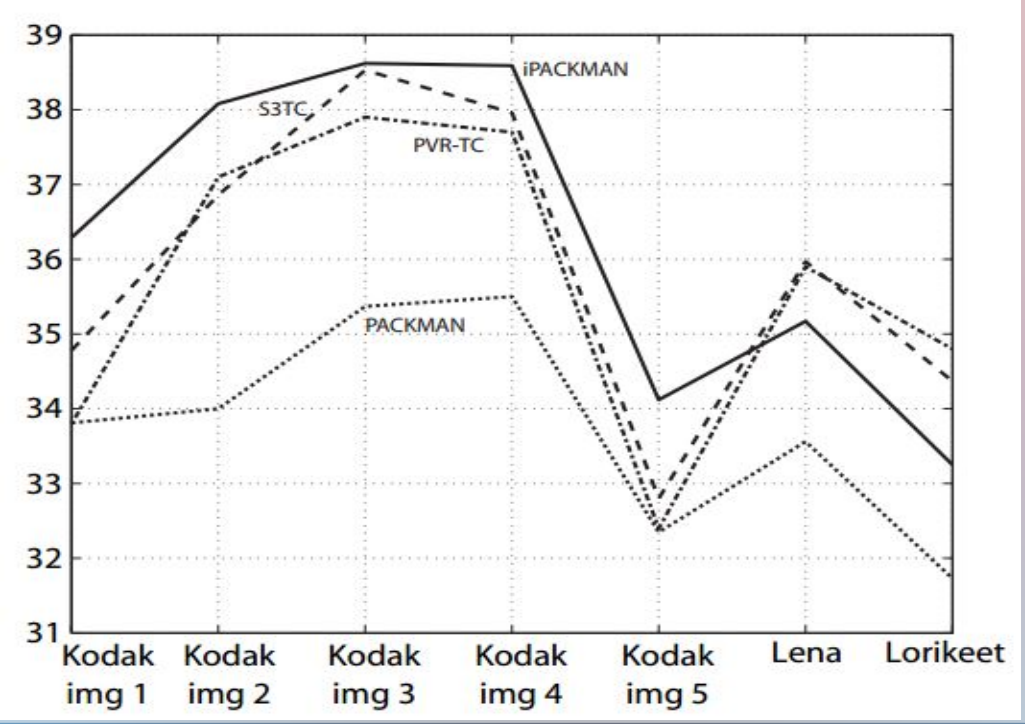
$$w_r^2 = 0{,}299 \; ; \; w_g^2 = 0{,}587 \; ; \; w_b^2 = 0{,}114$$

iPACKMAN - Im Vergleich (1/5)

- iPACKMAN wurde mit S3TC und der 4 Bit Version von PVR-TC verglichen
- S3TC:
  - Kodiert 4 x 4 Blöcke mit 64 Bits
  - Es besteht aus 5 Formaten (DXT1 – DXT5)
  - Unterscheiden sich in der Handhabung des alpha-Kanal
  - Es wurde in DirectX 6 aufgenommen
  - In DirectX 10 wurde es als veraltet deklariert
- PVR-TC:
  - Kodiert 4 x 4 Blöcke mit 64 Bits
  - Es existiert auch eine Codierung mit 2 x 2 Blöcken
  - Es wird in allen mobilen MAC-Produkten verwendet (iPad, iPad Touch, iPhone)
  - Nur theoretisch verglichen, da der Codec nicht öffentlich ist

iPACKMAN - Im Vergleich (2/5)

- Zum Vergleichen wurden 5 Bilder einer Testumgebung von Kodak verwendet und zwei weitere Bilder
- Um die Größe der Bilder zu normieren wurde die obere linke Ecke ausgewählt mit 512 x 512
- Keines der Bilder war zuvor eine Testumgebung

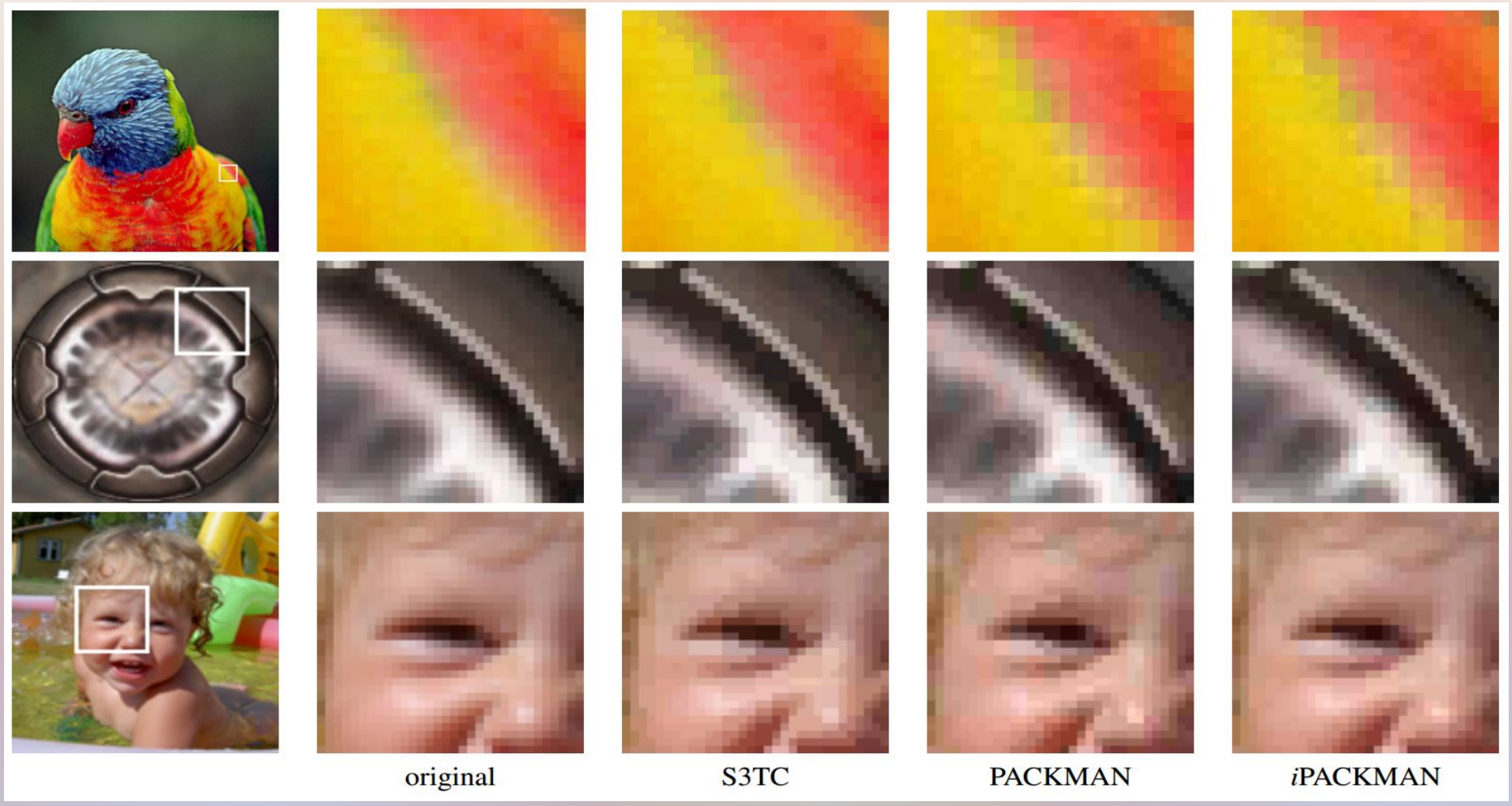


iPACKMAN - Im Vergleich (3/5)

	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	—

- In 5 der 7 Bilder war iPACKMAN stärker
- Es wurde die durchschnittliche Verbesserung angegeben
- 0,41dB stärker als S3TC und 0,65dB stärker als PVR-TC
- Die Community sagt, dass 0,25dB bereits einen sichtbaren Unterschied ausmacht

iPACKMAN - Im Vergleich (4/5)



- iPACKMAN ist schwächer bei Bildern mit fließenden Farbübergängen
- iPACKMAN ist stärker was die Leuchtdichte angeht, wegen der größeren Farbauswahl in der Grundfarbe

iPACKMAN - Im Vergleich (5/5)

- Einfarbiger Text auf farbigem Hintergrund sieht bei iPACKMAN gut aus
- Farbiger Text auf farbigem Hintergrund ist ungünstig für iPACKMAN

iPACKMAN - Transparenz

- Transparenz ist eine wichtige Eigenschaft in den Texturen
- Es werden 2 Ideen vorgestellt, da diese noch in der Entwicklung sind:
- Einfach:
  - Für jeden Pixel im Block werden 4 Bits kodiert
  - 4 x 4 x 4 = 64 Bits → 128 Bits pro 4 x 4 Block
  - Dies würde 16 verschiedene Stufen der Transparenz ermöglichen
- Gleiches Verfahren wie bei der Farbauswahl:
  - Man benötigt 2 Basisalpha-Werte pro 4 x 4 Block
  - Diese werden mit 8 Bit kodiert (5 + 3 oder 4 + 4)
  - Es gibt auch hier ein Flipbit und ein Diffbit und 2 x 3 Bits für die Tabellen
  - Die Tabellen haben aber 8 Einträge → 3 Bits pro Pixel statt 2
  - Damit liegt die Transparenz bei 64 Bits, die zu den 64 Bits Farbe hinzukommen (128 Bits)

iPACKMAN - Fazit

- Für mobile Geräte (Spielekonsolen oder Smartphones) ist wenig Bandbreite und Energiebedarf so wie möglichst wenig Platzverbrauch wichtig.
- iPACKMAN erfüllt diese Bedingungen
- Es ist im Durchschnitt stärker als die aktuellen Methoden
- Die Technik ist nicht limitiert auf mobile Geräte

- Aktuell wurden für RGB555 und RGB444 die gleichen Tabellen verwendet für die Leuchtdichte
- Die Farbübergänge von den RGB555 Blöcken ist nicht sonderlich groß
- Um iPACKMAN weiter zu verbessern kann man die Tabellen für Diffbit 1 und 0 unterscheiden und für RGB555 kleinere Werte auswählen

Vielen Dank für eure Aufmerksamkeit

Gibt es noch Fragen?