

# Media Data Compression

*Сжатие изображений*

**Дмитрий Ватолин**

*Московский Государственный Университет  
CS MSU Graphics&Media Lab*

**Часть вторая:  
СЖАТИЕ  
ИЗОБРАЖЕНИЙ**

# Благодарности



- ◆ Автор выражает признательность Александру Жиркову (Graphics&Media Lab) за помощь в подготовке этих лекций (разделы Jpeg-2000 и сжатие текстур).

# Сжатие изображений



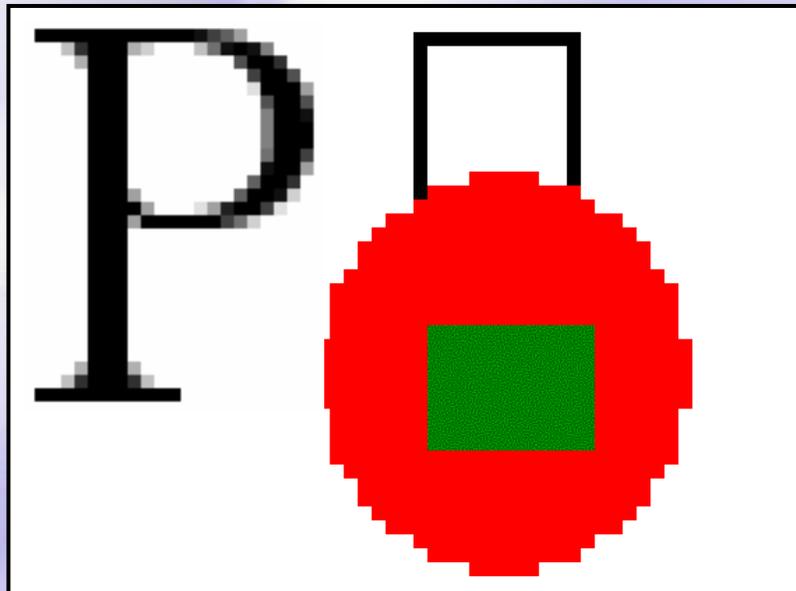
Будут рассмотрены алгоритмы:

- ◆ RLE
- ◆ LZW
- ◆ Хаффмана (CCITT G3)
- ◆ JPEG
- ◆ JPEG-2000
- ◆ фрактальный алгоритм

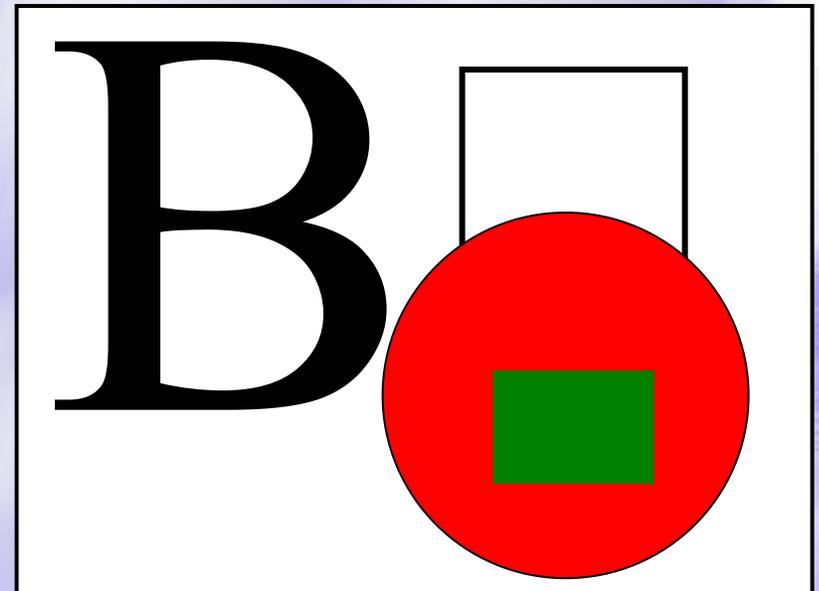
# Типы изображений

## Изображения

Растровые



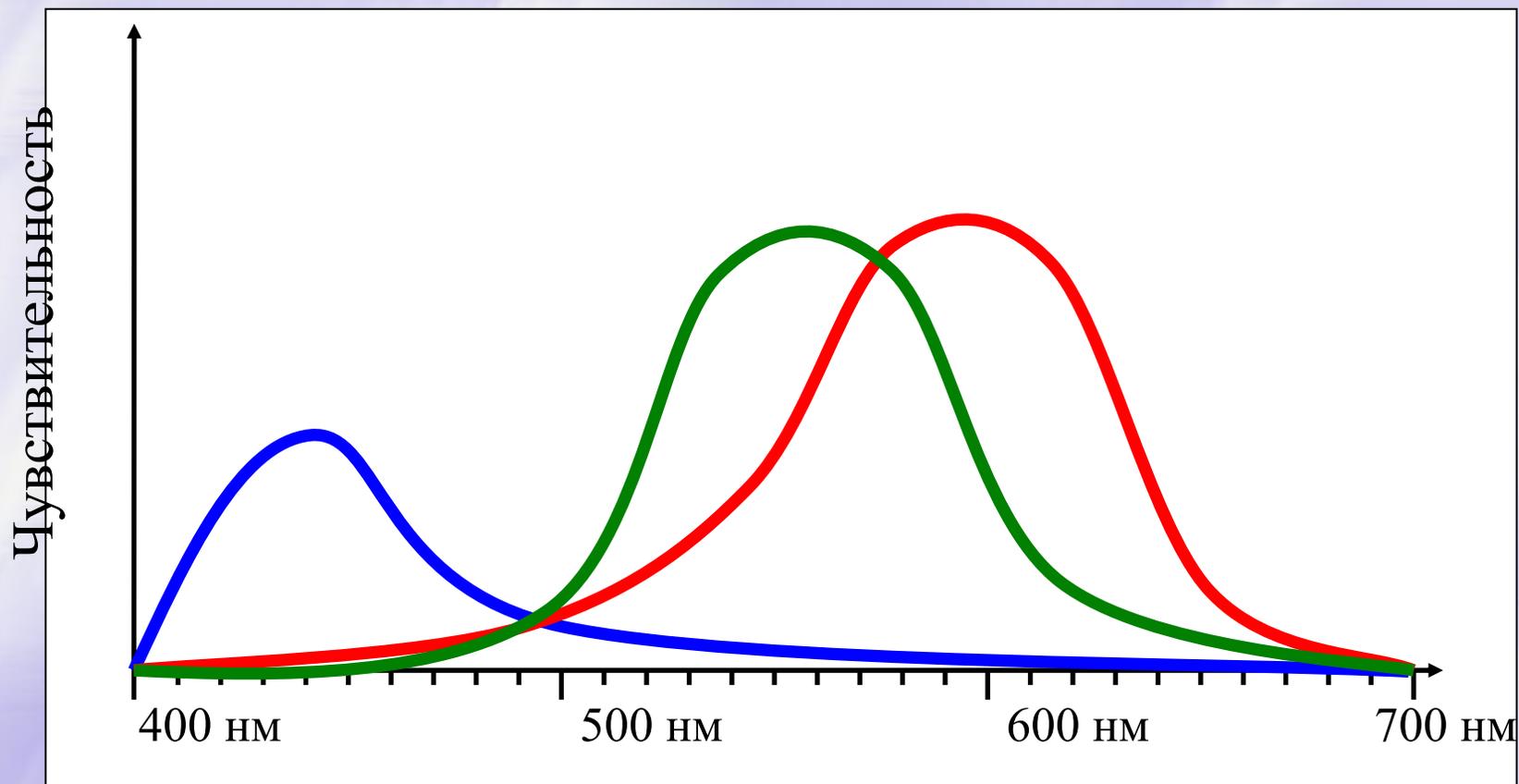
Векторные



# Типы изображений

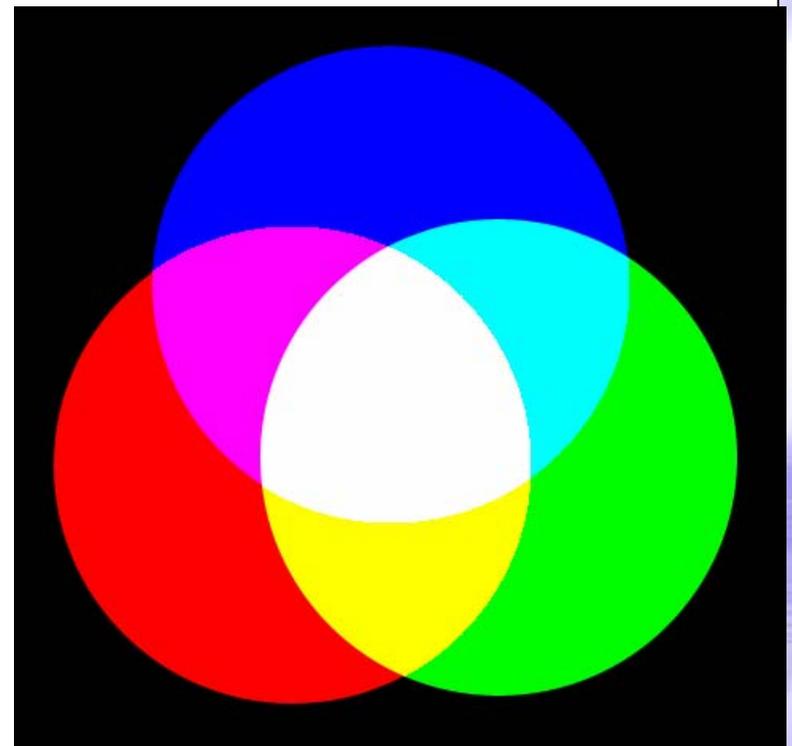
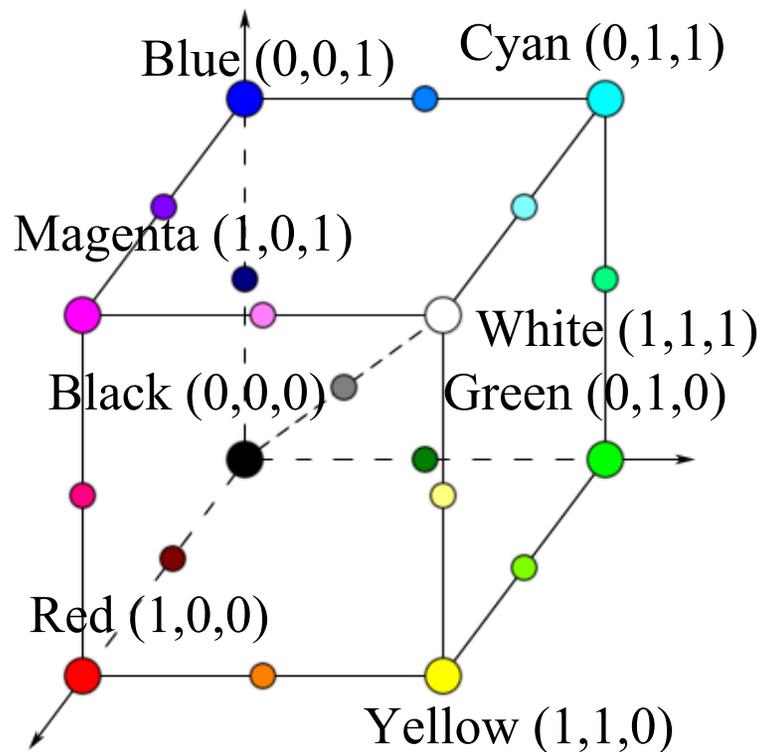
- ◆ Векторные
- ◆ Растровые
  - Палитровые
  - Безпалитровые
    - ◆ В системе цветопредставления  
RGB, CMYK, ...
    - ◆ В градациях серого

# Восприятие цвета



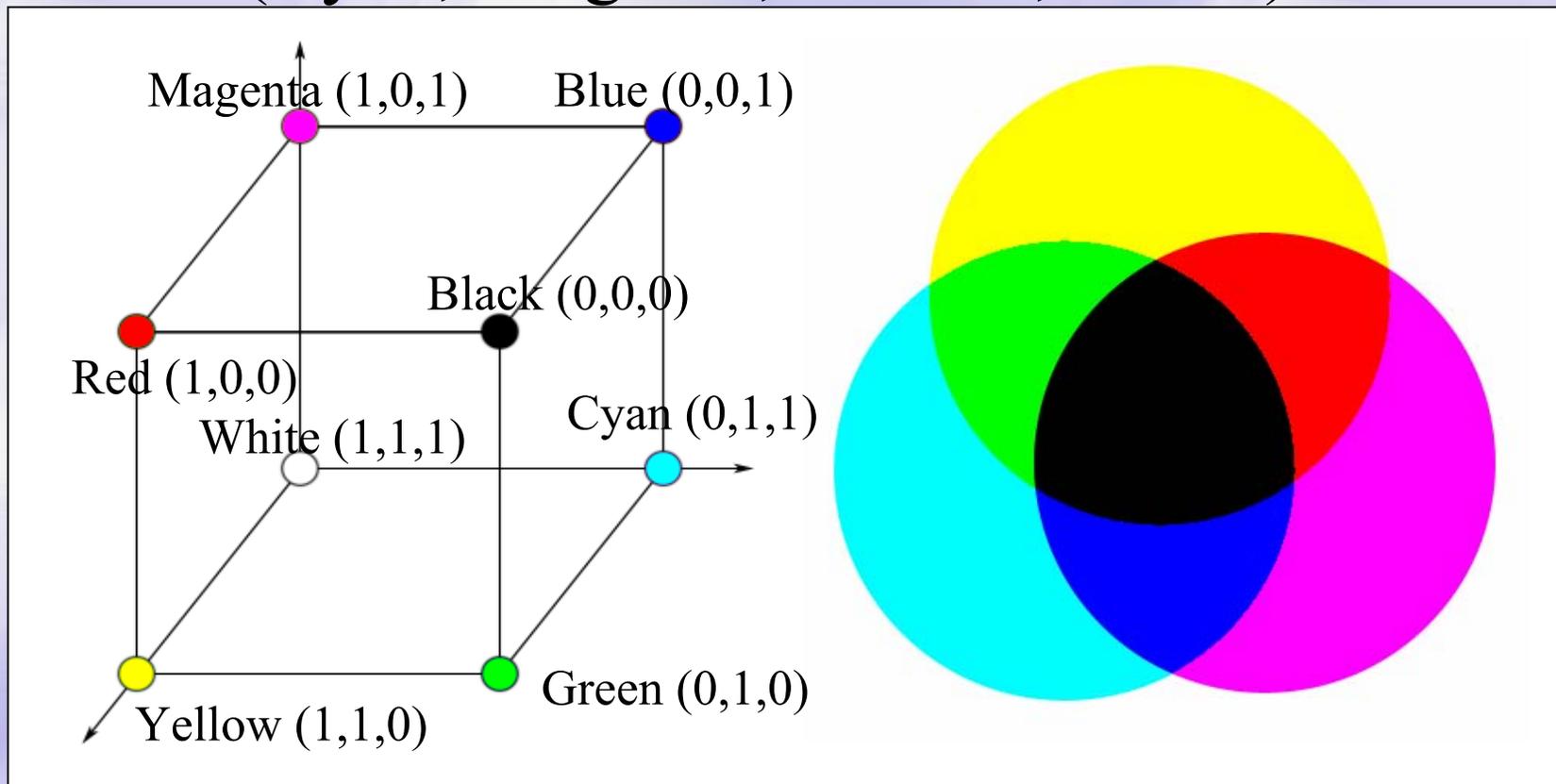
# Пространство RGB

## RGB (Red, Green, Blue)



# Пространство CMYK

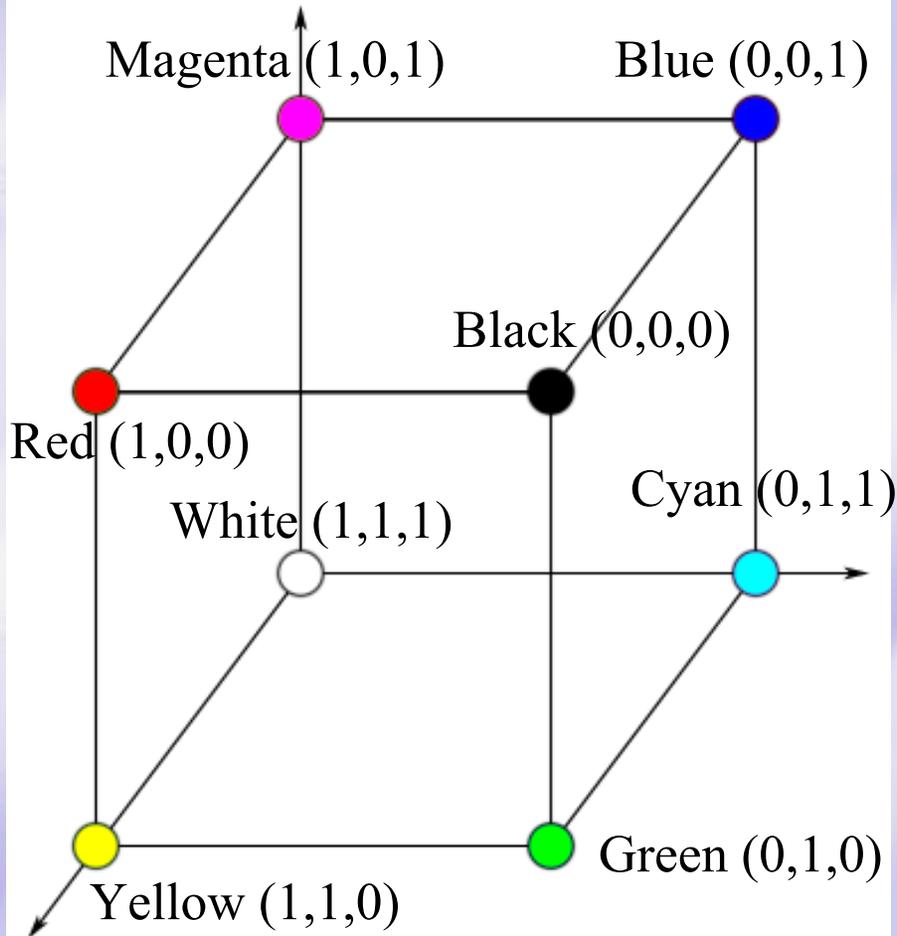
CMYK (Cyan, Magenta, Yellow, black).



# Расчет RGB, CMYK, CMY



GRAPHICS & MEDIA LAB  
VIDEO GROUP



RGB → CMY

$$C = 255 - R$$

$$M = 255 - G$$

$$Y = 255 - B.$$

CMY → CMYK

$$K = \min(C, M, Y),$$

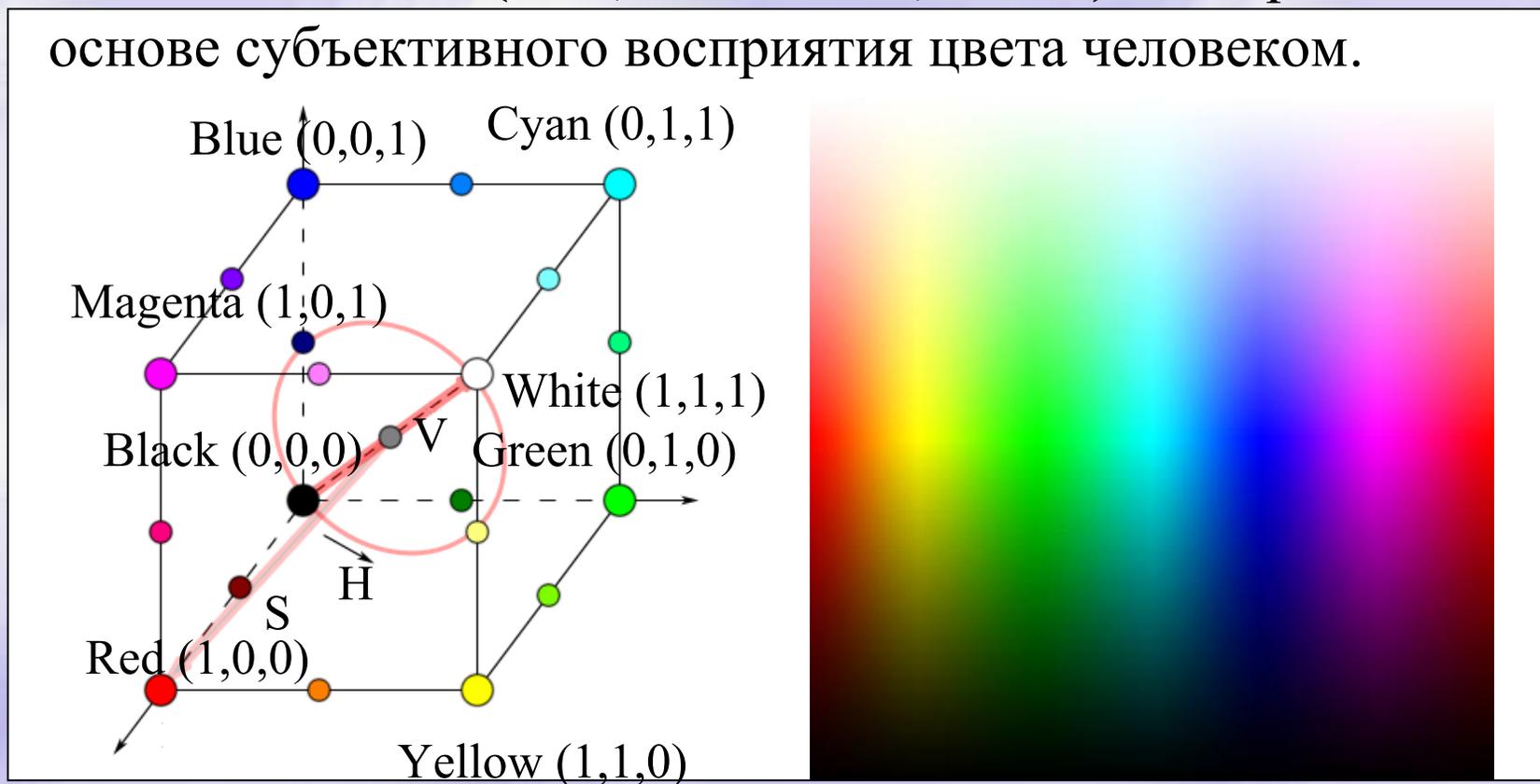
$$C = C - K,$$

$$M = M - K,$$

$$Y = Y - K.$$

# Пространство HSV

Модель **HSV (Hue, Saturation, Value)**. Построена на основе субъективного восприятия цвета человеком.



# Модель YUV



$$Y = 0.299R + 0.587G + 0,114B$$

$$U = -0.147R - 0.289G + 0,436B$$

$$V = 0.615R + 0.515G + 0,100B = 0,877(R - Y)$$

$$R = Y + 1.140V$$

$$G = Y - 0.395U - 0.581V$$

$$B = Y + 2.032U$$

# Модель YIQ



$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$I = 0.596 * R - 0.275 * G - 0.321 * B$$

$$Q = 0.212 * R - 0.523 * G + 0.311 * B$$

$$R = Y + 0.956 * I + 0.621 * Q$$

$$G = Y - 0.272 * I - 0.647 * Q$$

$$B = Y - 1.107 * I + 1.704 * Q$$

# Модель YCbCr (SDTV)

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.172 * R - 0.339 * G + 0.511 * B + 128$$

$$Cr = 0.511 * R - 0.428 * G + 0.083 * B + 128$$

$$R = Y + 1.371 ( Cr - 128 )$$

$$G = Y - 0.698 ( Cr - 128 ) - 0.336 ( Cb - 128 )$$

$$B = Y - 1.732 ( Cb - 128 )$$

# Классы изображений



- ◆ **Класс 1. Изображения с небольшим количеством цветов (4-16) и большими областями, заполненными одним цветом.** Плавные переходы цветов отсутствуют. Примеры: деловая графика — гистограммы, диаграммы, графики и т.п.
- ◆ **Класс 2. Изображения, с плавными переходами цветов, построенные на компьютере.** Примеры: графика презентаций, эскизные модели в САПР, изображения, построенные по методу Гуро.
- ◆ **Класс 3. Фотореалистичные изображения.** Пример: отсканированные фотографии.
- ◆ **Класс 4. Фотореалистичные изображения с наложением деловой графики.** Пример: реклама.

# Требования приложений к алгоритмам



- ◆ Высокая степень компрессии
- ◆ Высокое качество изображений
- ◆ Высокая скорость компрессии
- ◆ Высокая скорость декомпрессии
- ◆ Масштабирование изображений
- ◆ Возможность показать огрубленное изображение (низкого разрешения)
- ◆ Устойчивость к ошибкам
- ◆ Учет специфики изображения
- ◆ Редактируемость
- ◆ Небольшая стоимость аппаратной реализации.  
Эффективность программной реализации

# Критерии сравнения алгоритмов



Невозможно составить универсальное сравнительное описание известных алгоритмов.

- **Худший, средний и лучший коэффициенты сжатия.**
- **Класс изображений**
- **Симметричность**
- **Есть ли потери качества?**
- **Характерные особенности алгоритма**

# Алгоритм RLE

Данный алгоритм необычайно прост в реализации. Групповое кодирование — от английского Run Length Encoding (RLE). Изображение в нем вытягивается в цепочку байт по строкам раstra. Само сжатие в RLE происходит за счет того, что в исходном изображении встречаются цепочки одинаковых байт. Замена их на пары <счетчик повторений, значение> уменьшает избыточность данных.

# RLE – Первый вариант



```
Initialization(...);  
do {  
    byte = ImageFile.ReadNextByte();  
    if(является счетчиком(byte)) {  
        counter = Low6bits(byte)+1;  
        value = ImageFile.ReadNextByte();  
        for(i=1 to counter)  
            DecompressedFile.WriteByte(value)  
    }  
    else {  
        DecompressedFile.WriteByte(byte)  
    } while(ImageFile.EOF());
```

# RLE – Первый вариант (схема)



XX значение



11 6 бит

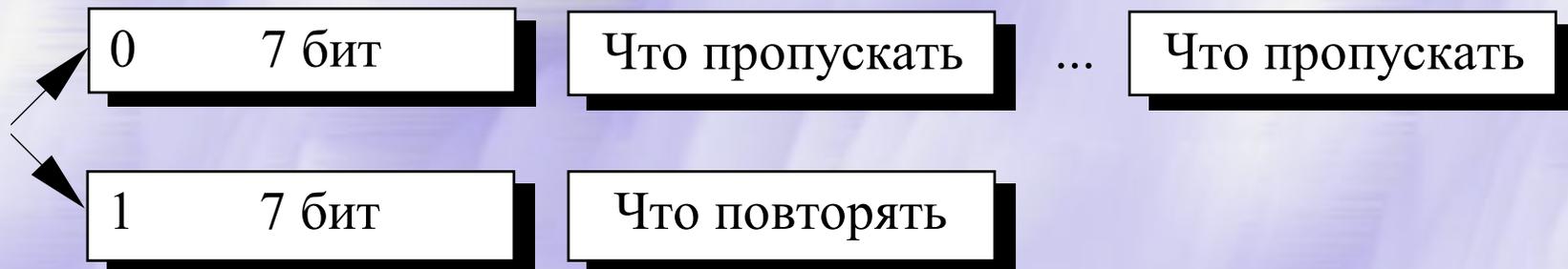
Что повторять

# RLE – Второй вариант



```
Initialization(...);  
do {  
    byte = ImageFile.ReadNextByte();  
    counter = Low7bits(byte)+1;  
    if(если признак повтора(byte)) {  
        value = ImageFile.ReadNextByte();  
        for (i=1 to counter)  
            CompressedFile.WriteByte(value)  
    }  
    else {  
        for(i=1 to counter){  
            value = ImageFile.ReadNextByte();  
            CompressedFile.WriteByte(value)  
        }  
        CompressedFile.WriteByte(byte)  
    } while (ImageFile.EOF());
```

# RLE – Схемы вариантов



# RLE – Характеристики



**Коэффициенты компрессии:** Первый вариант: 32, 2, 0,5. Второй вариант: 64, 3, 128/129. (Лучший, средний, худший коэффициенты)

**Класс изображений:** Ориентирован алгоритм на изображения с небольшим количеством цветов: деловую и научную графику.

**Симметричность:** Примерно единица.

**Характерные особенности:** К положительным сторонам алгоритма, пожалуй, можно отнести только то, что он не требует дополнительной памяти при архивации и разархивации, а также быстро работает. Интересная особенность группового кодирования состоит в том, что степень архивации для некоторых изображений может быть существенно повышена всего лишь за счет изменения порядка цветов в палитре изображения.

# Алгоритм LZW

Название алгоритм получил по первым буквам фамилий его разработчиков — Lempel, Ziv и Welch. Сжатие в нем, в отличие от RLE, осуществляется уже за счет **одинаковых цепочек** байт.

# Схема алгоритма LZ



# LZW / Сжатие



```
InitTable();
CompressedFile.WriteCode(ClearCode);
CurStr=пустая строка;

while(не ImageFile.EOF()){ //Пока не конец файла
    C=ImageFile.ReadNextByte();
    if(CurStr+C есть в таблице)
        CurStr=CurStr+C; //Приклеить символ к строке
    else {
        code=CodeForString(CurStr); //code-не байт!
        CompressedFile.WriteCode(code);
        AddStringToTable (CurStr+C);
        CurStr=C; // Строка из одного символа
    }
}
code=CodeForString(CurStr);
CompressedFile.WriteCode(code);
CompressedFile.WriteCode(CodeEndOfInformation);
```

# LZW / Пример



Пусть мы сжимаем последовательность 45, 55, 55, 151, 55, 55, 55.

“45” — есть в таблице;

“45, 55” — нет. Добавляем в таблицу <258>“45, 55”. В поток: <45>;

“55, 55” — нет. В таблицу: <259>“55, 55”. В поток: <55>;

“55, 151” — нет. В таблицу: <260>“55, 151”. В поток: <55>;

“151, 55” — нет. В таблицу: <261>“151, 55”. В поток: <151>;

“55, 55” — есть в таблице;

“55, 55, 55” — нет. В таблицу: “55, 55, 55” <262>. В поток: <259>;

Последовательность кодов для данного примера, попадающих в выходной поток: <256>, <45>, <55>, <55>, <151>, <259>.

# LZW / Добавление строк



Код этой строки добавляется в таблицу

$C_n, C_{n+1}, C_{n+2}, C_{n+3}, C_{n+4}, C_{n+5}, C_{n+6}, C_{n+7}, C_{n+8}, C_{n+9},$

Коды этих строк идут в выходной поток

# Таблица для LZW

0	'0'
...	...
255	'255'
256	ClearTable
257	EndOfInformation
258	
259	
...	
4095	

- Таблица состоит из 4096 строк.
- 256 и 257 являются служебными.
- 258 ... 4095 содержат непосредственно сжимаемую информацию.

# Пример – цепочка нулей

0	'0'
...	...
255	'255'
256	ClearTable
257	EndOfInformation
258	'00'
259	'000'
...	
4095	

Кол-во считываемых байт:

1                      2                      3

Общее число считанных байт:

1                      3                      6

Информация заносится в стр.:

-                      258                      259

# Степень сжатия цепочки нулей



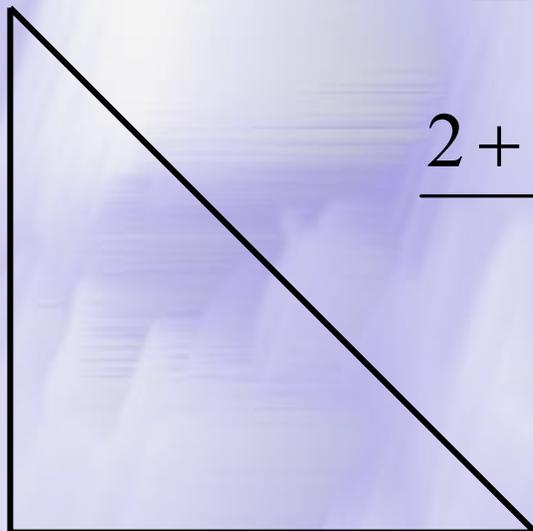
GRAPHICS & MEDIA LAB  
VIDEO GROUP

Рассчитываем арифметическую прогрессию:

$$S_n = \frac{a_1 + a_n}{2} * n$$

$$\frac{2 + 3839}{2} * 3839 = 7372799$$

0
..
255
256
257
258
259
..
4095



# Наихудший случай

0	'0'
...	...
255	'255'
256	ClearTable
257	EndOfInformation
258	'21'
259	'13'
...	
4095	

Последовательность :  
121314151617...

Мы видим, что у нас нет  
одинаковых цепочек  
даже из 2 символов =>  
сжатия не происходит.

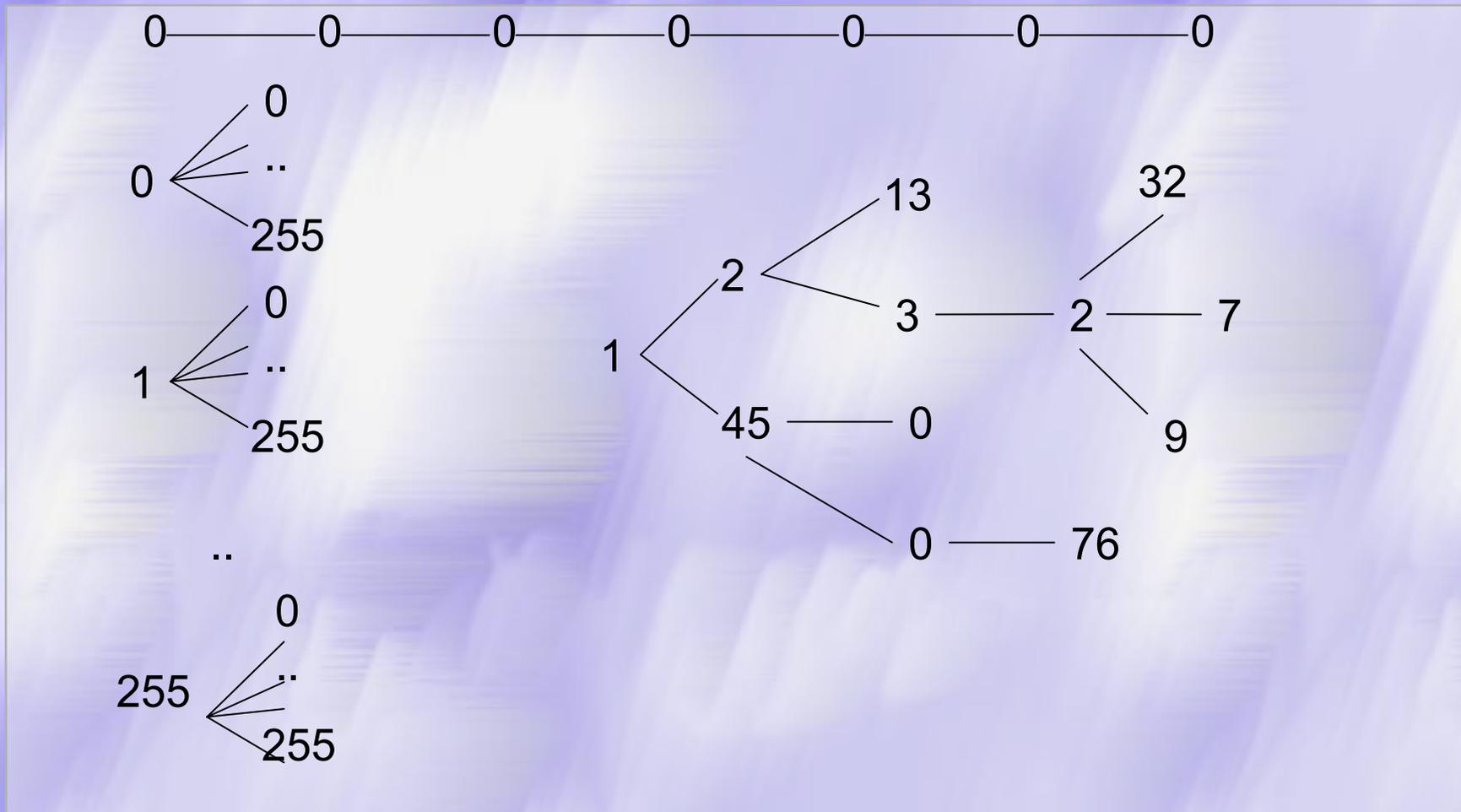
# Степень сжатия наихудшего случая



0	'0'
...	...
255	'255'
256	ClearTable
257	EndOfInformation
258	'21'
259	'13'
...	
4095	

- Происходит увеличение файла в 1.5 раза. Т.к. мы ни разу не встретили подстроку, которая уже есть в таблице.

# Таблица дерево



# Пример



Последовательность: 45, 55, 55, 151, 55, 55, 55.

“45” – есть в таблице;

“45, 55” – нет. В таблицу: <258>”45, 55”. В поток:<45>

“55, 55” – нет. В таблицу: <259>”55, 55”. В поток:<55>

“55, 151” – нет. В таблицу: <260>”55, 151”. В поток:<55>

“151, 55” – нет. В таблицу: <261>”151, 55”. В поток:<151>

“55, 55” – Есть в таблице;

“55, 55, 55” – нет. В таблицу: <262>”55, 55, 55”. В  
поток:<295>

Итого в потоке: <256>, <45>, <55>, <55>, <151>, <259>.

# Пример

0	'0'
...	...
255	'255'
256	ClearTable
257	EndOfInformation
258	'45, 55'
259	'55, 55'
260	'55, 151'
261	'151, 55'
262	'55, 55, 55'

Последовательность:  
45, 55, 55, 151, 55, 55, 55.

Итого в потоке:  
<256>, <45>, <55>, <55>,  
<151>, <259>.

# LZW / Декомпрессия



```
code=File.ReadCode();
while(code != CodeEndOfInformation){
    if(code = ClearCode) {
        InitTable();
        code=File.ReadCode();
        if(code = CodeEndOfInformation)
            {закончить работу};
        ImageFile.WriteString(StrFromTable(code));
        old_code=code;
    }
    else {
        if(InTable(code)) {
            ImageFile.WriteString(FromTable(code));
            AddStringToTable(StrFromTable(old_code)+
                FirstChar(StrFromTable(code)));
            old_code=code;
        }
        else {
            OutString= StrFromTable(old_code)+
                FirstChar(StrFromTable(old_code));
            ImageFile.WriteString(OutString);
            AddStringToTable(OutString);
            old_code=code;
        }
    }
}
```

# LZW / Характеристики

- **Коэффициенты компрессии:** Примерно 1000, 4, 5/7 (Лучший, средний, худший коэффициенты). Сжатие в 1000 раз достигается только на одноцветных изображениях размером кратным примерно 7 Мб.
- **Класс изображений:** Ориентирован LZW на 8-битные изображения, построенные на компьютере. Сжимает за счет одинаковых подцепочек в потоке.
- **Симметричность:** Почти симметричен, при условии оптимальной реализации операции поиска строки в таблице.

# Алгоритм Хаффмана

Использует только частоту появления одинаковых байт в изображении. Сопоставляет символам входного потока, которые встречаются большее число раз, цепочку бит меньшей длины. И, напротив, встречающимся редко — цепочку большей длины. Для сбора статистики требует двух проходов по изображению.



# Алгоритм Хаффмана-3

- ◆ **Коэффициенты компрессии:** 8, 1,5, 1 (Лучший, средний, худший коэффициенты).
- ◆ **Класс изображений:** Практически не применяется к изображениям в чистом виде. Обычно используется как один из этапов компрессии в более сложных схемах.
- ◆ **Симметричность:** 2 (за счет того, что требует двух проходов по массиву сжимаемых данных).
- ◆ **Характерные особенности:** Единственный алгоритм, который не увеличивает размера исходных данных в худшем случае (если не считать необходимости хранить таблицу перекодировки вместе с файлом).

# ССИТТ Group 3



Последовательности подряд идущих черных и белых точек в нем заменяются числом, равным их количеству. А этот ряд, уже в свою очередь, сжимается по Хаффману с фиксированной таблицей.



# Алгоритм ССІТТ G3



- ◆ Последовательности подряд идущих черных и белых точек заменяются числом, равным их количеству.
- ◆ Этот ряд сжимается по Хаффману с фиксированной таблицей.
- ◆ Каждая строка сжимается независимо, если строка начинается с черной точки, то считаем, что она начинается белой серией длиной 0. Например, последовательность длин серий 0, 3, 556, 10, ... означает, что в строке идут сначала 3 черных, 556 белых, 10 черных точек и т.д.

# Алгоритм компрессии:



```
For (по всем строкам изображения) {
    Преобразуем строку в набор длин серий;
    for (по всем сериям) {
        if (серия белая) {
            L = длина серии;
            while (L > 2623) { // 2623 = 2560 + 63
                L -= 2560; Записать белый код для (2560);
            }
            if (L > 63) {
                L2 = МаксимальныйСостКодМеньшеL(L);
                L -= L2; Записать белый код для (L2)
            };
            ЗаписатьБелыйКодДля(L); // код завершения
        } else {
            // аналогично для черных серий
            ...}
    }
}
```

# Пример работы алгоритма

В терминах регулярных выражений для каждой строки изображения выходной битовый поток вида:

$((\langle \text{Б-2560} \rangle)^*[\langle \text{Б-сст.} \rangle]\langle \text{Б-зв} \rangle(\langle \text{Ч-2560} \rangle)^*[\langle \text{Ч-сст} \rangle]\langle \text{Ч-зв} \rangle) + [(\langle \text{Б-2560} \rangle)^*[\langle \text{Б-сст.} \rangle]\langle \text{Б-зв.} \rangle]$ , где:

$()^*$  - повтор 0 или более раз,  $()^+$  - повтор 1 или более раз,  $[]$  – включение 1 или 0 раз.

Для примера 0, 3, 556, 10, ... , будет сформирован

код:  $\langle \text{Б-0} \rangle \langle \text{Ч-3} \rangle \langle \text{Б-512} \rangle \langle \text{Б-44} \rangle \langle \text{Ч-10} \rangle$  или

Согласно таблице:

00110101 10011001 01001011 010000100

Для приведенной строки в 569 бит полусен код длиной в 33 бита, т.е. Коэфф сжатия – 17 раз

# Таблица кодов завершения

Длина серии	Код белой подстроки	Код черной подстроки строки
0	00110101	0000110111
1	00111	010
2	0111	11
3	1000	10
4	1011	011
5	1100	0011
6	1110	0010
7	1111	00011

# Проблемы при сжатии



3. Identifying suitable the tasks and the methods to support them by the information systems (modeling an approach to using the LIS for the information systems) to be effective.

## 9. Summary and Future Research

In this paper, we have attempted to develop a general framework for studying the modeling of information systems for flexible manufacturing. The information modeling is considered with Petri net representations, including the input, output, and the information. In addition, the output requirements as an expert system for the framework are outlined. The framework can be used to describe functions, requirements of the information systems requirements in a general framework. Two model ways are the current and binary design level. A design is a stream of the expert system components, which is aimed to experiment with the current under study and as design and control in sub-iterations and the structure of the sub-iterations. The use of Petri nets allows consideration of possibility and flexibility of designing, simulating, and the ability to select a final specification of solutions to design and control in the information components of an EMS. Time and cost measures are considered in the information components of an EMS. Time and cost measures that can be used in control and sub-iterations are given. The expert system can be used to analyze the status performance of various configurations of the EMS information system under study.

In the future, we will consider making an effort on building a prototype of a general framework as an interactive computer system body a system and also representation with EMS information systems design issues.

## References

- 1. Chang, T. S. and Wang, J. S. (1991) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 2. Chang, T. S. and Wang, J. S. (1992) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 3. Chang, T. S. and Wang, J. S. (1993) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 4. Chang, T. S. and Wang, J. S. (1994) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 5. Chang, T. S. and Wang, J. S. (1995) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 6. Chang, T. S. and Wang, J. S. (1996) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 7. Chang, T. S. and Wang, J. S. (1997) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 8. Chang, T. S. and Wang, J. S. (1998) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 9. Chang, T. S. and Wang, J. S. (1999) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.
- 10. Chang, T. S. and Wang, J. S. (2000) Information Systems Design: A Case Study in Manufacturing. In: *Information Systems Design: A Case Study in Manufacturing* (Ed. by T. S. Chang and J. S. Wang), pp. 1-10. Addison-Wesley, Reading, MA.

## Book Reviews

The Journal Book Review Journal. The American Book Review, 1997, vol. 10, no. 1.

U.S. Copyright Law. Boston, 1997, vol. 10, no. 1.

The book is concerned with legal programs. It provides information on the book review, an internal study, a set of required software. The ANI is the theory of computer logic systems, where the information is used to imply the sequential objects, sub-iterations of the program.

The book is concerned with legal programs. It provides information on the book review, an internal study, a set of required software. The ANI is the theory of computer logic systems, where the information is used to imply the sequential objects, sub-iterations of the program.

Пример, когда часть страницы идет под косым углом + разворот книги ТЕМНЫЙ

# Проблемы при сжатии

Пример факса  
(часть текста  
рекомендаций  
стандарта  
ССИТТ) на  
японском (?)  
языке.

かつ、勧告を行なうことができる。」(同  
と第188号にいわれる「意見」とは、  
語では、「勧告(Recommendation)」とよ  
国際法的には強制力をもたないものであ  
等各国を拘束する力をもっているもの  
的分野では、電信規則のごとき、各国  
をもたないので、実際にある機器の仕様  
されたこの「意見」に従わなければ、同  
が多い。この意見(または勧告)は、同  
ついて、具体的意見を表明するもので、  
半自動化しようとする場合、その信号

# CSITT Group 3 / Характеристики



- ◆ **Коэффициенты компрессии:** лучший коэффициент стремится в пределе к 213.(3), средний 2, в худшем случае увеличивает файл в 5 раз.
- ◆ **Класс изображений:** Двухцветные черно-белые изображения, в которых преобладают большие пространства, заполненные белым цветом.
- ◆ **Симметричность:** Близка к 1.
- ◆ **Характерные особенности:** Данный алгоритм чрезвычайно прост в реализации, быстр и может быть легко реализован аппаратно.

# Сжатие изображений с потерями

# Качество изображений



**Не существует метода оценки  
качества изображения, полностью  
адекватного человеческому  
восприятию**

# PSNR

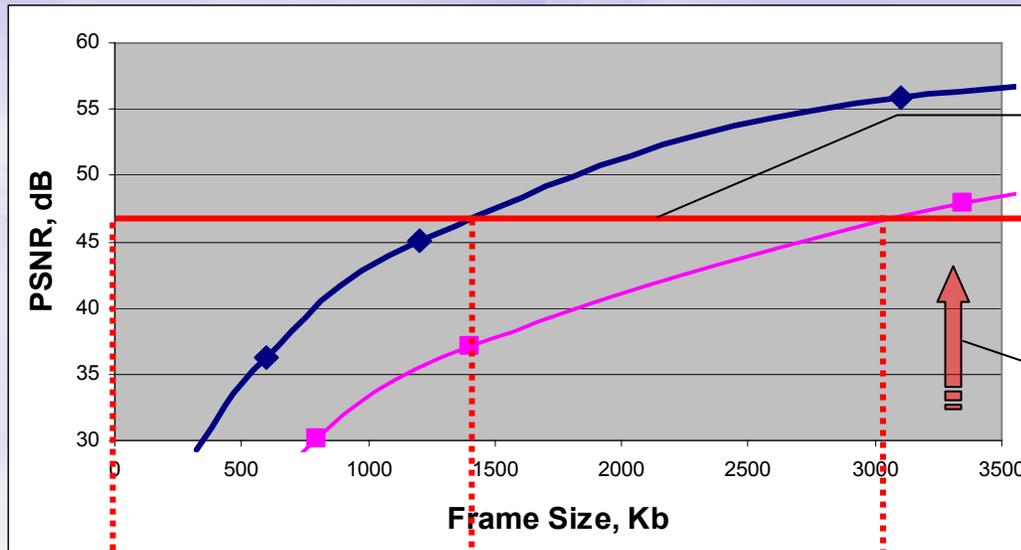


Базовые метрики –  
Y-PSNR, U-PSNR, V-PSNR

$$d(x,y) = 10 \cdot \log_{10} \frac{255^2 \cdot n^2}{\sum_{i=1, j=1}^{n, n} (x_{ij} - y_{ij})^2}$$

Хорошо работают только на высоком  
качестве.

# Как интерпретировать PSNR



Линия  
одинакового  
качества

Чем выше, тем  
лучше

Разные размеры кадров для разных алгоритмов

Преимущество для синей линии

# Тестовое изображение «Барбара»



Много полосок  
(высоких  
частот) в  
разных  
направлениях и  
разной  
толщины

# Тестовое изображение «Boat»



Много тонких  
деталей и  
наклонных  
границ в  
разном  
направлении

# Задача тестовых наборов

## Основные задачи тестовых наборов

- ◆ Обеспечить единую базу сравнения разных алгоритмов (в статьях и т.п.)
- ◆ Обеспечить выявление разных типов артефактов в алгоритмах

# Алгоритм JPEG



Алгоритм разработан в 1991 году группой экспертов в области фотографии (JPEG — Joint Photographic Expert Group — подразделение в рамках ISO) специально для сжатия 24-битных изображений.

Алгоритм основан на дискретном косинусном преобразовании (в дальнейшем ДКП), применяемом к матрице изображения для получения некоторой новой матрицы коэффициентов.

# Алгоритм JPEG / RGB в YUV



Изначально при сжатии изображение переводится в цветное пространство YUV. Упрощенно перевод можно представить с помощью матрицы перехода:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.4187 & -0.0813 \\ 0.1687 & -0.3313 & 0.5 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{pmatrix} * \begin{pmatrix} \begin{pmatrix} Y \\ U \\ V \end{pmatrix} - \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \end{pmatrix}$$

# Алгоритм JPEG



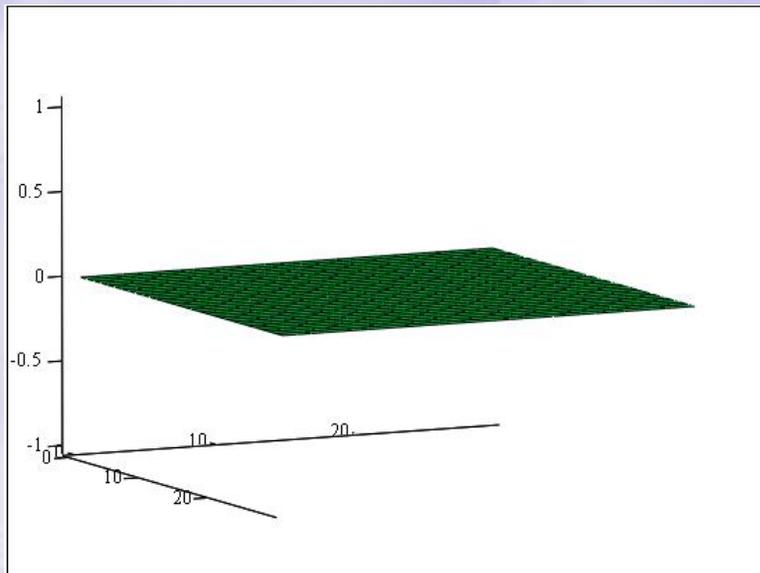
$$Y[u,v] = \frac{1}{4} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C(i,u) \times C(j,v) \times y[i,j]$$

где  $C(i,u) = A(u) \times \cos\left(\frac{(2 \times i + 1) \times u \times \pi}{2 \cdot n}\right)$

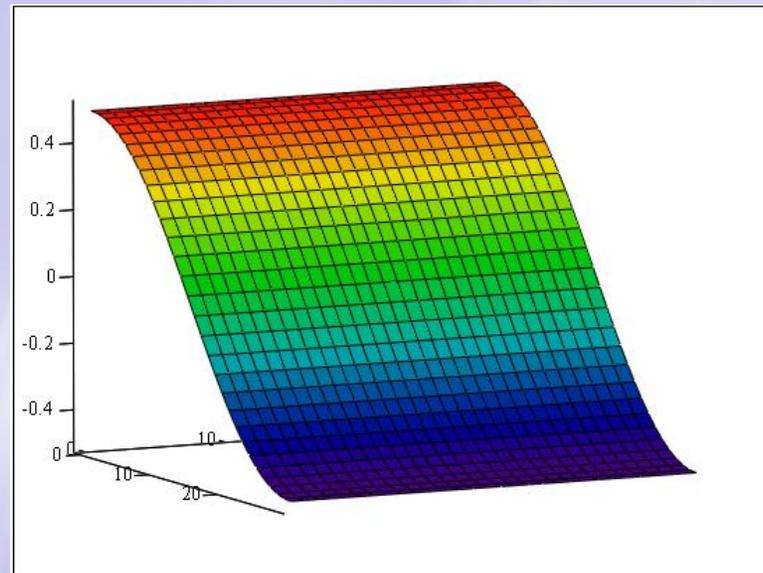
$$A(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{for } u \equiv 0 \\ 1, & \text{for } u \neq 0 \end{cases}$$

a <sub>0,0</sub>	a <sub>0,1</sub>	a <sub>0,2</sub>	a <sub>0,3</sub>	a <sub>0,4</sub>	a <sub>0,5</sub>	a <sub>0,6</sub>	a <sub>0,7</sub>
a <sub>1,0</sub>	a <sub>1,1</sub>	a <sub>1,2</sub>	a <sub>1,3</sub>	a <sub>1,4</sub>	a <sub>1,5</sub>	a <sub>1,6</sub>	a <sub>1,7</sub>
a <sub>2,0</sub>	a <sub>2,1</sub>	a <sub>2,2</sub>	a <sub>2,3</sub>	a <sub>3,0</sub>			
a <sub>3,0</sub>	a <sub>3,0</sub>	a <sub>3,0</sub>	a <sub>3,0</sub>				
a <sub>4,0</sub>	a <sub>4,1</sub>	a <sub>4,2</sub>					
a <sub>5,0</sub>	a <sub>5,1</sub>						
a <sub>6,0</sub>	a <sub>6,1</sub>						
a <sub>7,0</sub>	a <sub>7,1</sub>						

# Алгоритм JPEG / Примеры DCT

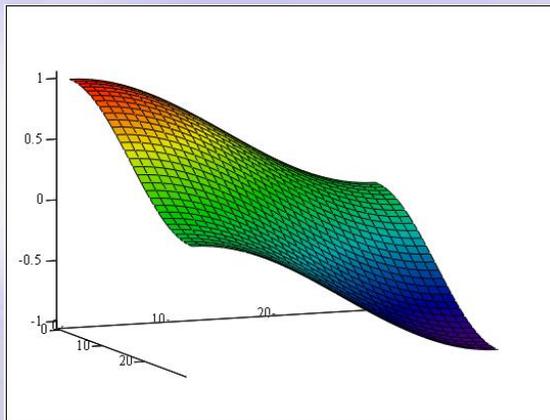
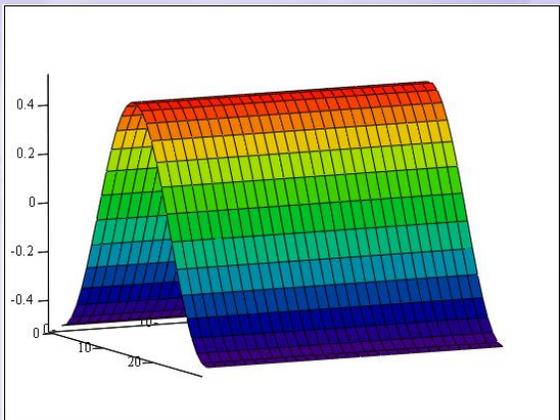


$$f(x, y) := \frac{0}{2}$$



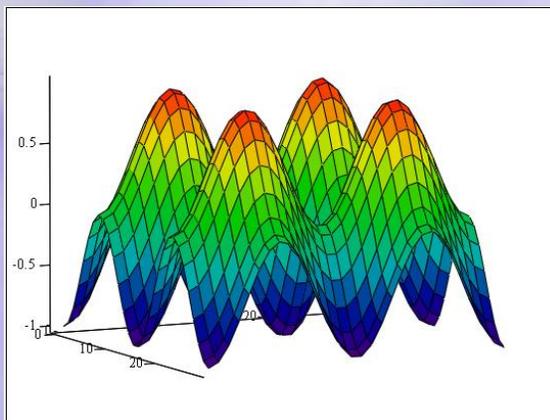
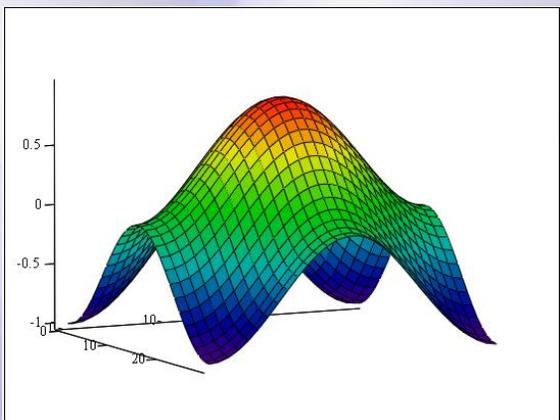
$$f(x, y) := \frac{-\cos(x)}{2}$$

# Алгоритм JPEG / Примеры DCT



$$f(x, y) := \frac{-\cos(2x)}{2}$$

$$f(x, y) := \frac{-\cos(y) - \cos(x)}{2}$$



$$f(x, y) := \frac{-\cos(2y) - \cos(2x)}{2}$$

$$f(x, y) := \frac{-\cos(4y) - \cos(4x)}{2}$$

# Алгоритм JPEG / Характеристики



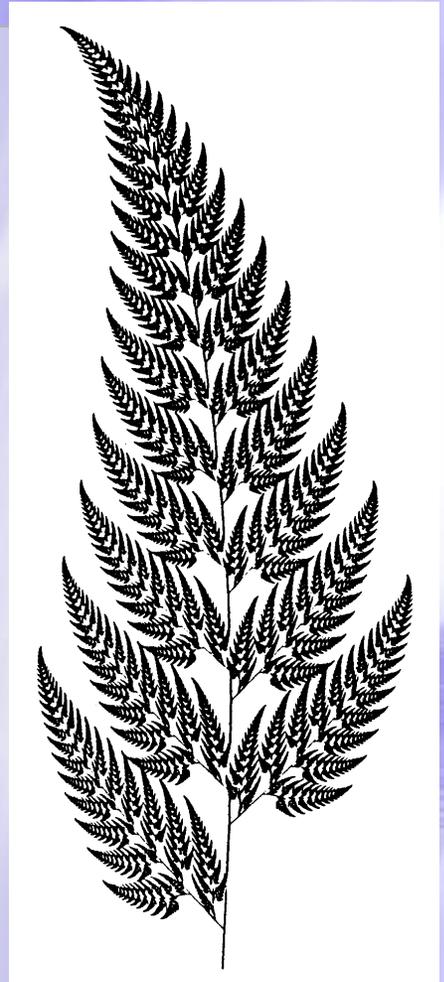
- ◆ **Коэффициенты компрессии:** 2-100 (Задается пользователем).
- ◆ **Класс изображений:** Полноцветные 24 битные изображения или изображения в градациях серого без резких переходов цветов (фотографии).
- ◆ **Симметричность:** 1
- ◆ **Характерные особенности:** В некоторых случаях, алгоритм создает “ореол” вокруг резких горизонтальных и вертикальных границ в изображении (эффект Гиббса). Кроме того, при высокой степени сжатия изображение распадается на блоки 8x8 пикселей.

# Фрактальное сжатие



Фрактальная компрессия —  
алгоритм с потерей  
информации, появившийся в  
1992 году

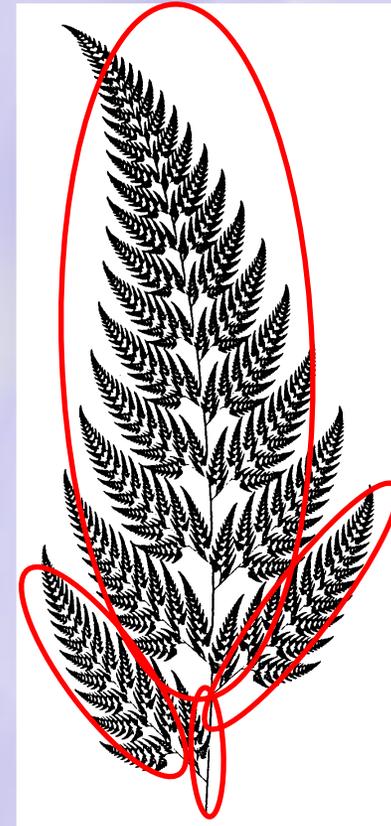
Он использует **аффинные преобразования** для построения изображений, что позволяет очень компактно задавать сложные структуры.



# Пример самоподобия

Папоротник Барнсли  
Состоит задается четырьмя  
аффинными преобразованиями

Изображение имеет четыре области,  
каждая из которых подобна  
изображению, и их объединение  
покрывает все изображение.  
(Стебель, Листья.)

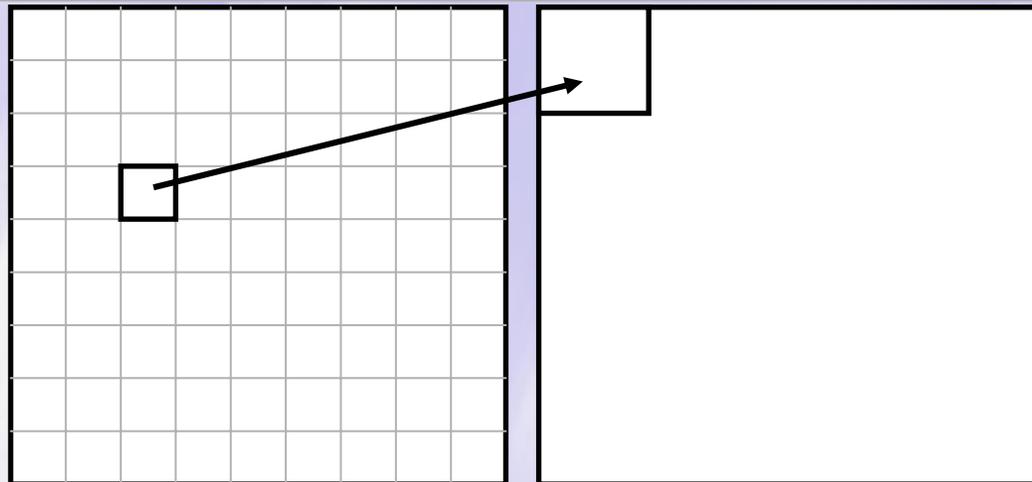


# Идея фрактального алгоритма



Сжатие осуществляется за счет поиска самоподобных участков в изображении

# Идея фрактального алгоритма



$$w_i(\bar{x}) = w_i \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & p \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} e \\ f \\ q \end{pmatrix}$$

Для перевода участков один в другой используется аффинное преобразование

# Аффинное преобразование



**Определение.** Преобразование  $w: R^2 \rightarrow R^2$ , представимое в виде

$$w(\vec{o}) = w \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

где  $a, b, c, d, e, f$  действительные числа и называется двумерным аффинным преобразованием.

**Определение.** Преобразование  $w: R^3 \rightarrow R^3$ , представимое в виде

$$w(\vec{o}) = w \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a & b & t \\ c & d & u \\ r & s & p \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} e \\ f \\ q \end{pmatrix}$$

где  $a, b, c, d, e, f, p, q, r, s, t, u$  действительные числа и называется трехмерным аффинным преобразованием.

# Аттрактор и теорема о сжимающем преобразовании



**Определение.** Пусть  $f : X \rightarrow X$  — преобразование в пространстве  $X$ . Точка  $x_f \in X$  такая, что  $f(x_f) = x_f$  называется неподвижной точкой (аттрактором) преобразования.

**Определение.** Преобразование  $f : X \rightarrow X$  в метрическом пространстве  $(X, d)$  называется сжимающим, если существует число  $s$ :  $0 \leq s < 1$ , такое, что  $d(f(x), f(y)) \leq s \cdot d(x, y) \quad \forall x, y \in X$

**Теорема. (О сжимающем преобразовании)**

Пусть  $f : X \rightarrow X$  — сжимающее преобразование в полном метрическом пространстве  $(X, d)$ . Тогда существует в точности одна неподвижная точка  $x_f \in X$  этого преобразования, и для любой точки  $x \in X$  последовательность  $\{f^n(x) : n = 0, 1, 2, \dots\}$  сходится к  $x_f$ .

# Изображение и IFS

**Определение.** Изображением называется функция  $S$ , определенная на единичном квадрате и принимающая значения от 0 до 1 или

$$S(x, y) \in [0...1] \quad \forall x, y \in [0...1]$$

**Определение.** Конечная совокупность  $W$  сжимающих трехмерных аффинных преобразований, определенных на областях, таких, что  $w_i(D_i) = R_i$  и  $R_i \cap R_j = \emptyset \quad \forall i \neq j$  называется системой итерируемых функций (IFS).

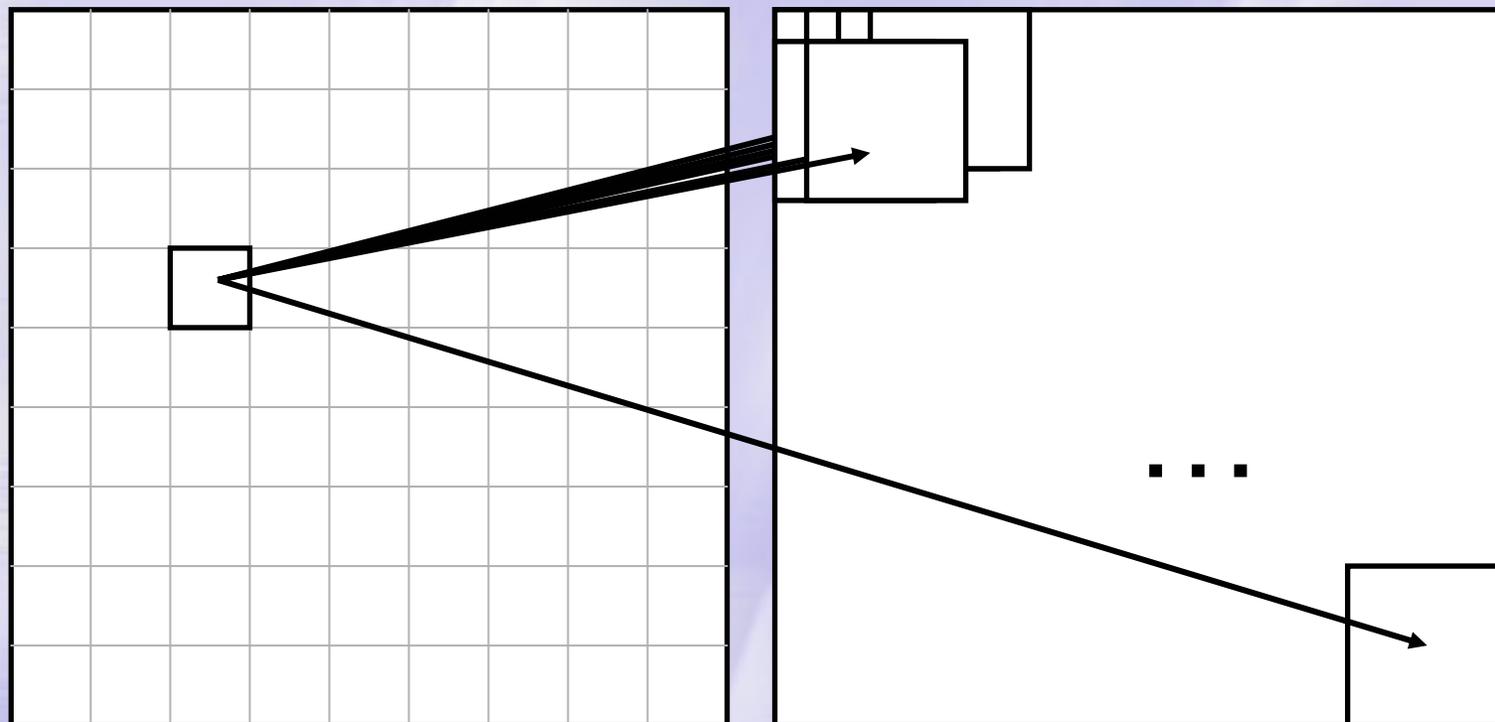
# Идея фрактального алгоритма



- ◆ Мы записываем в файл коэффициенты
- ◆ Если размер коэффициентов меньше размера исходного файла, мы получаем алгоритм сжатия

**Существенный недостаток — большое время сжатия. Т.е. на сжатие рисунка уходят часы на мощных компьютерах.**

# Поиск соответствий



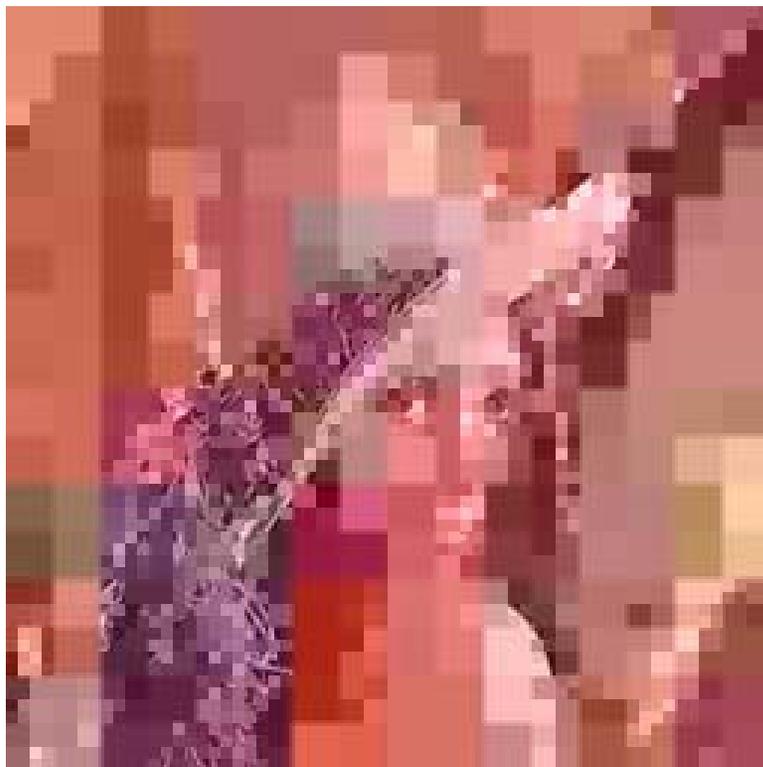
# Декомпрессор



Читаем из файла коэффициенты всех блоков,  
и создаем изображение нужного размера  
(обычно черного цвета)

```
Until(Изображение не перестанет изменяться) {  
    For(every range (R)) {  
        D=image->CopyBlock(D_coord_for_R);  
        For(every pixel(i,j) in the block{  
             $R_{ij} = 0.75D_{ij} + oR;$   
        } //Next pixel  
    } //Next block  
} //Until end
```

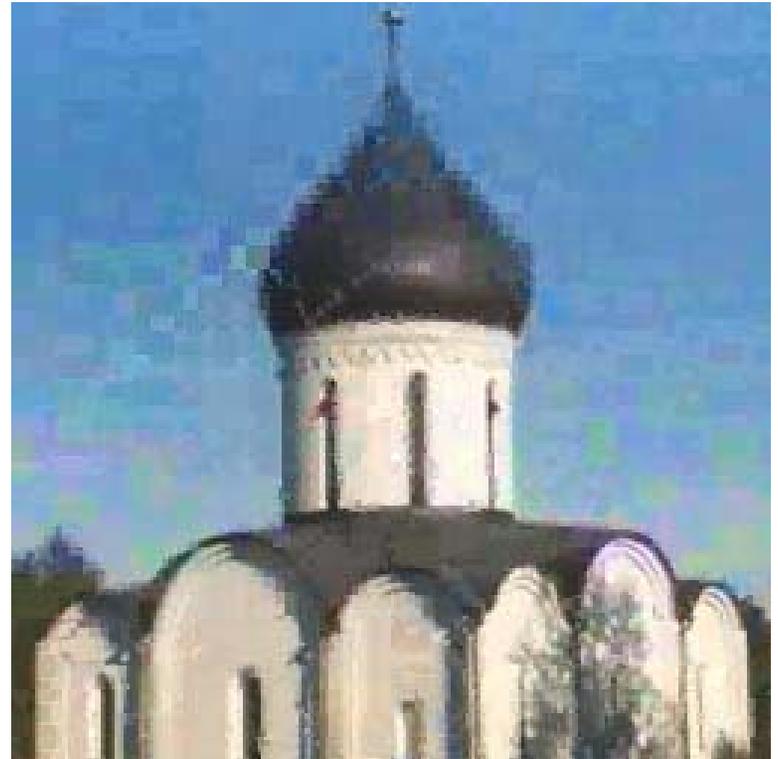
# Декомпрессия: Шаг 1



# Декомпрессия: Шаг 2



# Декомпрессия: Шаг 3



# Декомпрессия: Шаг 4



# Декомпрессия: Шаг 5

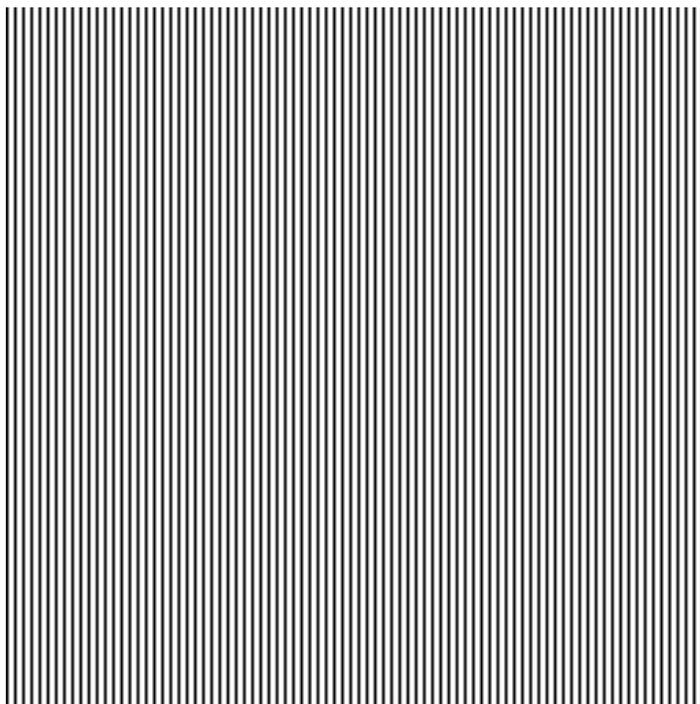


# Примеры восстановления

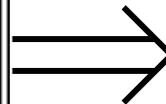
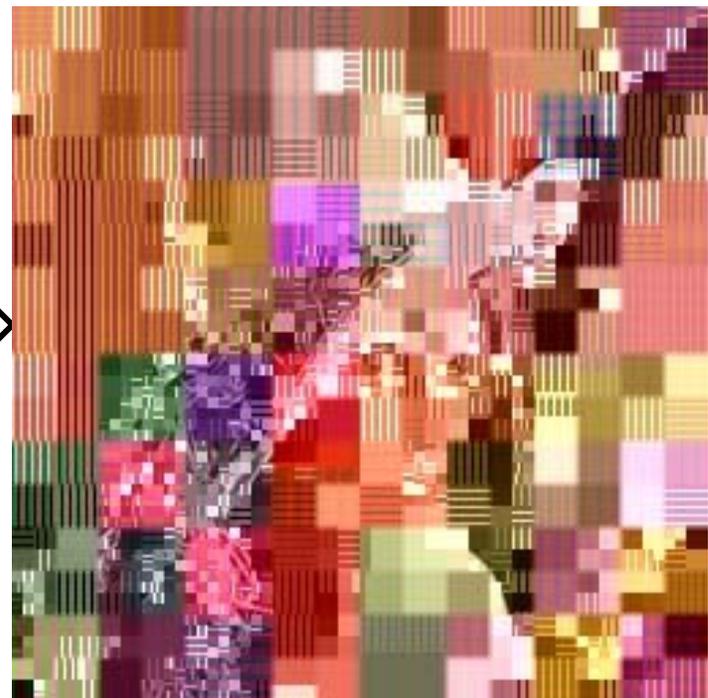


# Пример восстановления

Исходное изображение



Первый шаг восстановления



# Фрактальное сжатие / Характеристики



- **Коэффициенты компрессии:** От 2 до 100 раз.
- **Класс изображений:** 24-битные и 8-битные grayscale изображения.
- **Симметричность:** Существенно несимметричен. Коэффициент несимметричности достигает 10000.

# **СЖАТИЕ ИЗОБРАЖЕНИЙ**

*JPEG-2000*

*Сравнение с JPEG*

# JPEG 2000



Алгоритм JPEG 2000 разработан той же группой экспертов в области фотографии, что и JPEG. Формирование JPEG как международного стандарта было закончено в 1992 году. В 1997 стало ясно, что необходим новый, более гибкий и мощный стандарт, который и был доработан к зиме 2000 года.

# JPEG 2000 /

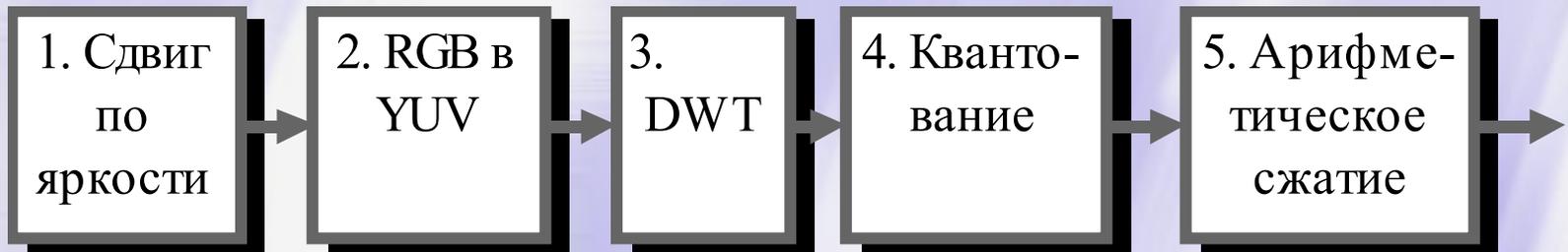
## Идея алгоритма



Базовая схема JPEG-2000 очень похожа на базовую схему JPEG. Отличия заключаются в следующем:

- 1) Вместо дискретного косинусного преобразования (DCT) используется дискретное вэйвлет-преобразование (DWT).
- 2) Вместо кодирования по Хаффману используется арифметическое сжатие.
- 3) В алгоритм изначально заложено управление качеством областей изображения.
- 4) Не используется уменьшение разрешения цветоразностных компонент U и V.
- 5) Кодирование с явным заданием требуемого размера на ряду с традиционным метод кодирования по качеству.
- 6) Поддержка сжатия без потерь. Поддержка сжатия однобитных (2-цветных) изображений
- 7) На уровне формата поддерживается прозрачность.

# JPEG 2000 / Схема



Конвейер операций, используемый в JPEG-2000

# JPEG 2000 / RGB в YUV



Этот шаг аналогичен JPEG (см. матрицы преобразования в описании JPEG), за тем исключением, что кроме преобразования с потерями предусмотрено также и преобразование без потерь.

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} \left[ \frac{R + 2G + B}{4} \right] \\ R - G \\ B - G \end{pmatrix}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} Y \\ Y - \left[ \frac{U + V}{4} \right] \\ V + G \end{pmatrix}$$

# JPEG 2000 / DWT



В одномерном случае

$$y_{output}(2n) = \sum_{j=0}^{N-1} x_{input}(j) \cdot h_H(j - 2n)$$

применение

DWT – это

«обычная фильтрация».

$$y_{output}(2n + 1) = \sum_{j=0}^{N-1} x_{input}(j) \cdot h_L(j - 2n - 1)$$

Из строки  $x$  мы получаем строку  $y$  по приведенным формулам.

В двумерном случае мы сначала применяем эти формулы по всем строкам изображения, а потом по всем столбцам.

# JPEG 2000 / DWT коэффициенты



## Коэффициенты при упаковке

<b>i</b>	<b>Низкочастотные коэффициенты <math>h_L(i)</math></b>	<b>Высокочастотные коэффициенты <math>h_H(i)</math></b>
<b>0</b>	<b>1.115087052456994</b>	<b>0.6029490182363579</b>
<b>±1</b>	<b>0.5912717631142470</b>	<b>-0.2668641184428723</b>
<b>±2</b>	<b>-0.05754352622849957</b>	<b>-0.07822326652898785</b>
<b>±3</b>	<b>-0.09127176311424948</b>	<b>0.01686411844287495</b>
<b>±4</b>	<b>0</b>	<b>0.02674875741080976</b>
<b>Другие i</b>	<b>0</b>	<b>0</b>

коэффициенты '9/7' DWT при сжатии с потерями

# JPEG 2000 /

# DWT коэффициенты (без потерь)



i	При упаковке		При распаковке	
	Низкочастотные коэффициенты $h_L(i)$	Высокочастотные коэффициенты $h_H(i)$	Низкочастотные коэффициенты $g_L(i)$	Высокочастотные коэффициенты $g_H(i)$
0	6/8	1	1	6/8
$\pm 1$	2/8	-1/2	1/2	-2/8
$\pm 2$	-1/8	0	0	-1/8

Коэффициенты '5/3' DWT при сжатии без потерь

# JPEG 2000 / DWT без потерь



Поскольку большинство  $h_L(i)$ , кроме окрестности  $i=0$ , равны 0, то можно переписать приведенные формулы короче:

$$y_{out}(2n) = \frac{-x_{in}(2n-1) + 2 \cdot x_{in}(2n) + 6 \cdot x_{in}(2n+1) + 2 \cdot x_{in}(2n+2) - x_{in}(2n+3)}{8}$$

$$y_{out}(2n+1) = -\frac{x_{in}(2n)}{2} + x_{in}(2n+1) - \frac{x_{in}(2n+2)}{2}$$

А потом еще и упростить, как:

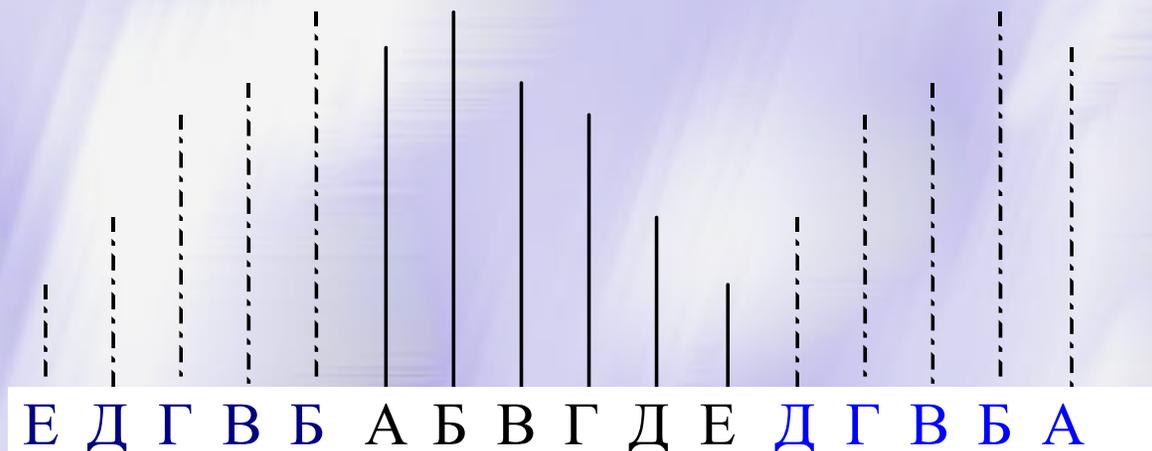
$$y_{out}(2n+1) = x_{in}(2n+1) - \left[ \frac{x_{in}(2n) + x_{in}(2n+2)}{2} \right]$$

$$y_{out}(2n) = x_{in}(2n) + \left[ \frac{y_{out}(2n-1) + y_{out}(2n+1) + 2}{4} \right]$$

# JPEG 2000 / DWT – края



Применение DWT на краях изображения:



Симметричное расширение изображения (яркости АБ...Е) по строке вправо и влево

# JPEG 2000 / DWT – Пример



Пусть мы преобразуем строку из 10 пикселей. Расширим ее значения вправо и влево и применим DWT преобразование:

n	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
$x_{in}$	3	2	1	2	3	7	10	15	12	9	10	5	10	9
$y_{out}$		0	1	0	3	1	11	4	13	-2	8	-5		

Получившаяся строка 1, 0, 3, 1, 11, 4, 13, -2, 8, -5 полностью и однозначно задает исходные данные. Обратное преобразование осуществляется по:

$$x_{out}(2n) = y_{out}(2n) - \left[ \frac{y_{out}(2n-1) + y_{out}(2n+1) + 2}{4} \right]$$

$$x_{out}(2n+1) = y_{out}(2n+1) + \left[ \frac{x_{out}(2n) + x_{out}(2n+2)}{2} \right]$$

n	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
$y_{out}$		0	1	0	3	1	11	4	13	-2	8	-5	8	-2
$x_{out}$			1	2	3	7	10	15	12	9	10	5	10	

# JPEG 2000 /

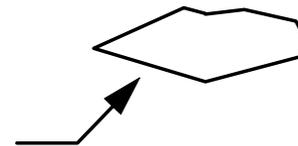
## Изменение качества областей



Изображение,  
сжатое с  
большими  
потерями



Области  
повышенного  
качества

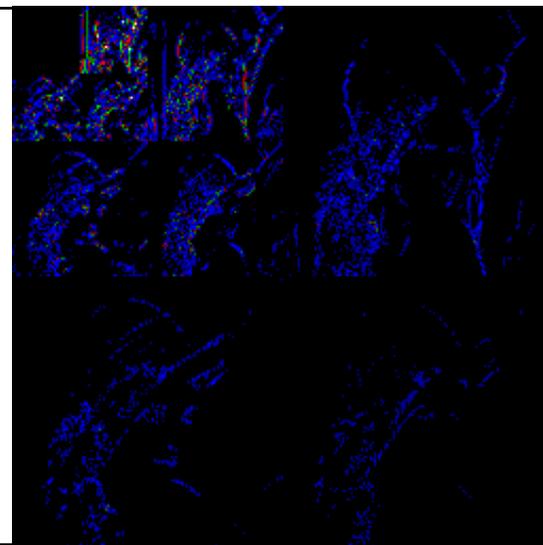
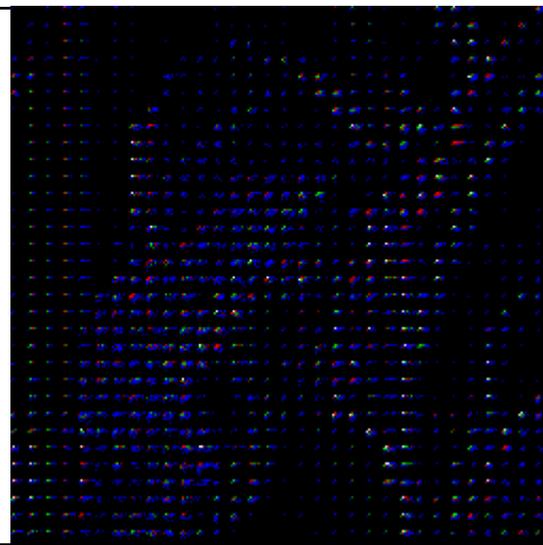
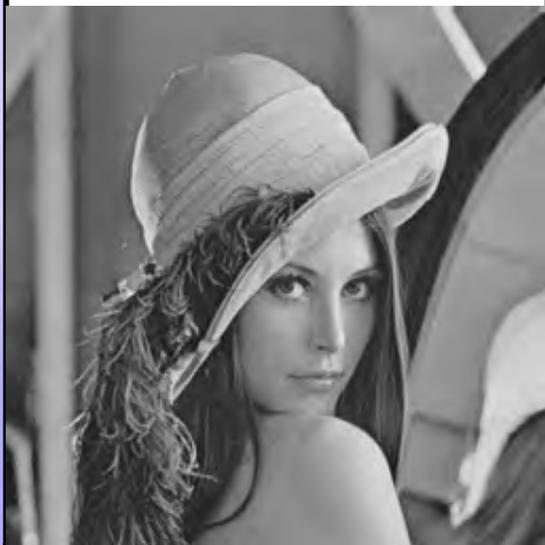


Когда практически достигнут предел сжатия изображения в целом и различные методы дают очень небольшой выигрыш, мы можем существенно (в разы) увеличить степень сжатия за счет изменения качества разных участков изображения.

# Сравнение этапа сжатия без потерь JPEG и JPEG-2000



	JPEG	JPEG-2000
Структура разбиения	пространственная	частотная
Проход по коэффициентам	частотный	пространственный
Кодирование коэффициентов	групповое кодирование не нулевых	проходы по соответствующим битам коэффициентов



# JPEG-2000 / Кодирование битовых плоскостей



- ◆ Разбиение DWT-пространства на одинаковые блоки, по умолчанию размером 64x64
  - Каждый блок кодируется независимо от других
  - В отличие от EZW и SPIHT (set partitioning in hierarchical trees) межуровневые зависимости не учитываются
- ◆ Кодирование одной битовой плоскости одного блока осуществляется в три этапа:
  - Кодирование старших бит
  - Уточняющий проход
  - Очищающий проход

# JPEG-2000 / Кодирование битовых плоскостей



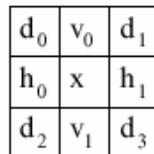
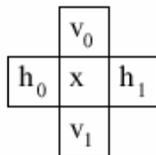
- ◆ Для каждого прохода используется бинарное адаптивное арифметическое кодирование и контекстное моделирование:
  - **Арифметическое** кодирование позволяет кодировать символы с произвольным распределением вероятности (не только равных степени двойки как у таблиц Хаффмана)
  - **Адаптивность** позволяет задавать распределение вероятностей исходя из статистики уже закодированных данных
  - **Контекстное моделирование** позволяет использовать закономерности между и внутри потоков данных, путем использования различных вероятностных таблиц для разных ‘контекстов’

# JPEG-2000 / Кодирование битовых плоскостей



## ◆ Кодирование старших бит

- Кодирование предсказанных и при подтверждении гипотезы, кодирование знака
- Контекст при кодирования значимости:
  - ◆ значимость соседних 8-ми связанных коэффициентов
  - ◆ Тип бэнда: LL, LH, HL, HH
- Контекст при кодирования знака:
  - ◆ Значимость и знаки 4-х связанных коэффициентов



4-х и 8-ми  
СВЯЗНОСТЬ

# JPEG-2000 / Кодирование битовых плоскостей

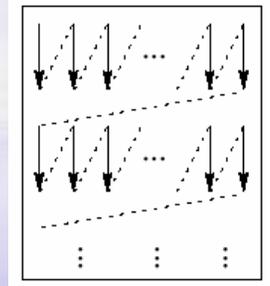


## ◆ Уточняющий проход:

- Кодирование существенных битов расположенных ниже первого
- Контекст для бита:
  - ◆ ‘Это второй по важности бит?’
  - ◆ Значимость 8-ми связанных коэффициентов

## ◆ Очищающий проход:

- Кодирование не предсказанных, но существенных битов



Порядок обхода

# JPEG-2000 /

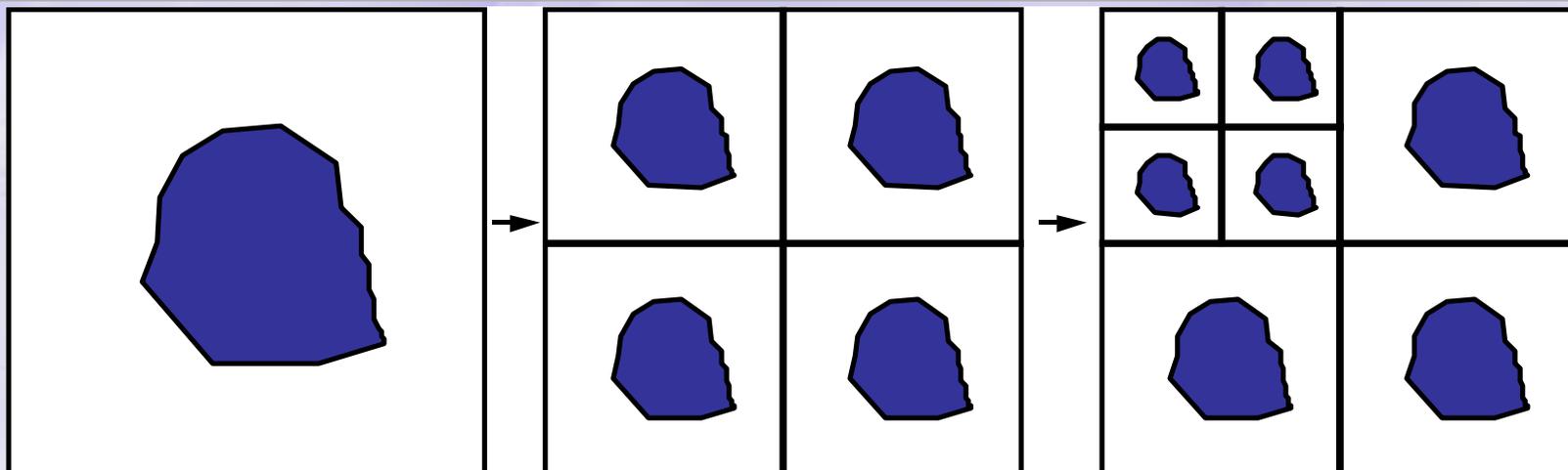
## Кодирование: Внешний цикл



- ◆ Цель: записать в поток результаты кодирования битовых плоскостей
  - Единица потока – пакет. Пакет – компрессированный проход одной битовой плоскости одного блока
  - Сортировка пакетов в соответствии с выбранной стратегией:
    - ◆ Слой-разрешение-компонента-позиция: возможность прогрессивной визуализации
    - ◆ Разрешение-слой-компонента-позиция: прогрессивная восстановление по разрешению
    - ◆ Другие три сценария

# JPEG-2000 /

## Изменение качества областей



В JPEG-2000 используется неявное представление бинарной маски, внутри которой точность квантования коэффициентов другая нежели вне её. Метод представления и компрессии маски будет описан позже.

# JPEG-2000 / Алгоритм изменения качества областей



- ◆ Изменение качества выделенных областей
  - При кодировании:
    - ◆ Разделение битовой маски на выделенные и принадлежащие фону
    - ◆ Достаточный сдвиг (умножение на степень двойки) выделенных коэффициентов на  $N$ , что бы биты выделенного изображения и фона не пересекались
  - При декодировании:
    - ◆ После распаковки, все коэффициенты большие  $2^N$  сдвигаются направо на  $N$
  - Плюсы такого подхода:
    - ◆ Нет необходимости явного хранения бинарной маски

# JPEG-2000 / Пресеты КВАНТОВАНИЯ



**Адаптированный пресет,  
лучше качество**

**Стандартный пресет,  
больше PSNR**

# JPEG / JPEG-2000: ‘Лена’



Сравнение JPEG & JPEG-2000 при сжатии в 30 раз

# JPEG / JPEG-2000: Сжатие в 130 раз



JPEG: сохранено больше деталей



JPEG-2000: отсутствие блочных артефактов

# Алгоритм JPEG-2000 / Характеристики



- ◆ **Коэффициенты компрессии:** 2-200 (Задается пользователем), возможно сжатие без потерь.
- ◆ **Класс изображений:** Полноцветные 24-битные изображения, изображения в градациях серого, 1-битные изображения (JPEG-2000 - наиболее универсален).
- ◆ **Симметричность:** 1
- ◆ **Характерные особенности:** Можно задавать качество участков изображений.

# Сравнение алгоритмов (1)



Алгоритм	За счет чего происходит сжатие
<b>RLE</b>	2 2 2 2 2 2 2 15 15 15 — Подряд идущие цвета
<b>LZW</b>	2 3 15 40 2 3 15 40 — Одинаковые подцепочки
<b>Хаффмана</b>	2 2 3 2 2 4 3 2 2 2 4 — Разная частота появления цвета
<b>Wavelet</b>	Плавные переходы цветов и отсутствие резких границ
<b>JPEG</b>	Отсутствие резких границ
<b>JPEG-2000</b>	Плавные переходы цветов и отсутствие резких границ
<b>Фрактальный</b>	Подобие между элементами изображения

# Сравнение алгоритмов (2)



Алгоритм	К-ты сжатия	Сим-метрич-ность	Не что ориенти рован	Поте ри	Разме рност ь
<b>RLE</b>	1/32 1/2 2/1	1	3-8 bit	Нет	1D
<b>LZW</b>	1/1000 1/4 7/5	1.2-3	1-8 bit	Нет	1D
<b>Хаффмана</b>	1/8 2/3 1/1	1-1.5	5-8 bit	Нет	1D
<b>JBIG</b>	1.5	~1	1-bit.	Нет	2D
<b>Lossless JPEG</b>	2	~1	24-bit greyscale	Нет	2D
<b>JPEG</b>	2-20	~1	24-bits greyscale	Да	2D
<b>JPEG-2000</b>	2-200	1.5	24-bits greyscale	Да	2D
<b>Fractal</b>	2-2000	1000-10000	24-bits greyscale	Да	2D

# СЖАТИЕ ТЕКСТУР

*Специфика*

*Обзор форматов S3TC, FXT1, CD,  
CTF-8, CTF-12*

# Компрессия текстур: Специфика



## Требования:

- Прямой доступ к пикселям (текселям) из сжатого представления
- Эффективность аппаратной реализации

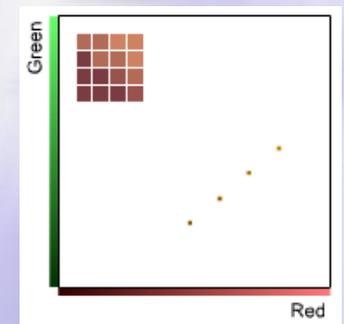
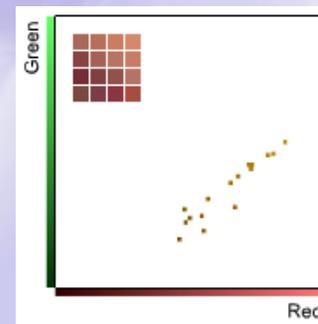
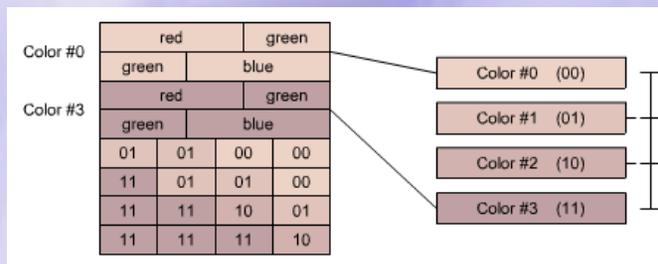
## Распространенный подход:

- Блочная компрессия с локальной палитризацией
- Фиксированный коэффициент сжатия

# Компрессия текстур: Алгоритм S3TC\*

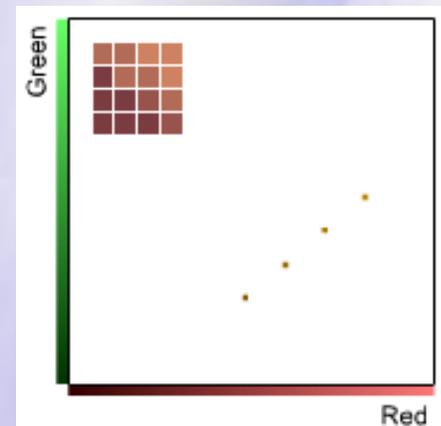
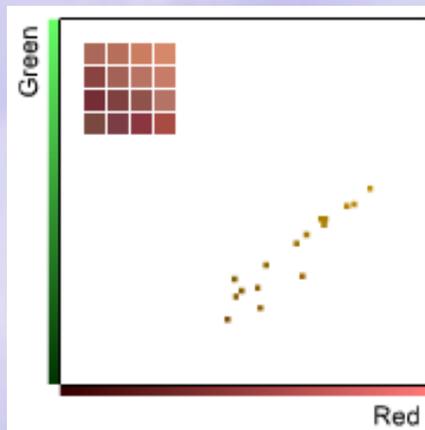
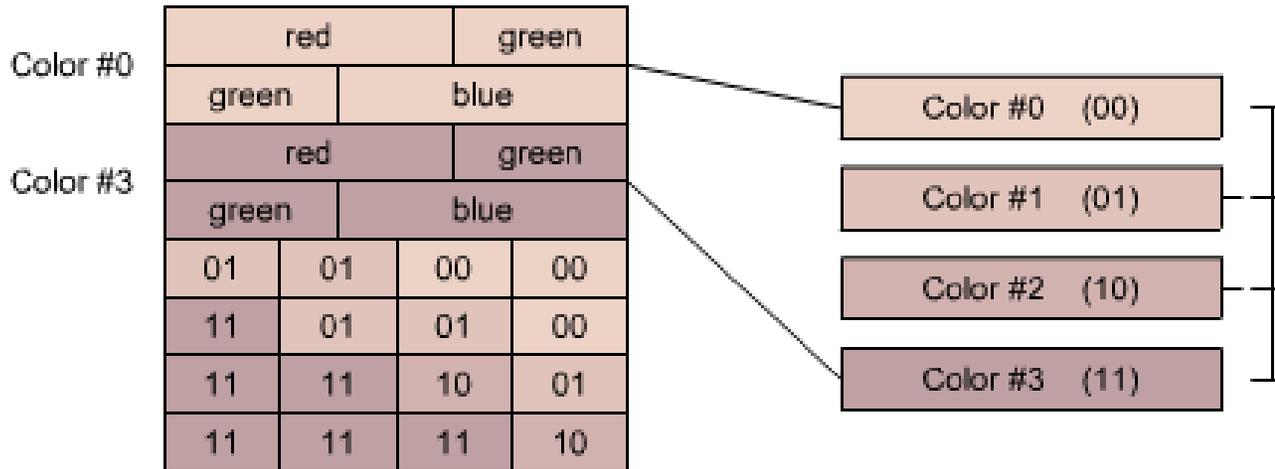
## Идея:

- Четыре цвета на блок 4x4, но хранения только двух базовых, остальные линейно интерполируются
- Преимущества:
  - ◆ Шесть раз сжатие; достаточное качество; простой для аппаратной реализации алгоритм; стандарт де-факто
  - ◆ Хранении базовых цветов в 16 битном формате, но возможно использование всех 16 млн



\*) SONICblue (originally S3 Inc.)

# Компрессия текстур: Алгоритм S3TC\*



# Компрессия текстур: Формат FXT1\*



## Идея/Цель

- Улучшение S3TC (альфа-канал, больше блок, несколько адаптивных алгоритмов)

## Алгоритм

- Для каждого блока 4x8 используется один из 4-х методов сжатия:
  - ◆ MIXED: 2 бита/индекс, по два базовых цвета на подблок 4x4, 1 интерполируется между ними, 1 прозрачный
  - ◆ HI: 3 бита/индекс, 2 базовых, 5 интерполируются, 1 прозрачный
  - ◆ CHROMA: 2 бита/индекс, 4 базовых цвета
  - ◆ ALPHA: 2 бита/индекс, 2 цвета по 20 бит (RGBA)

\*)3dfx Iterative Inc.

# Компрессия текстур: Оценка формата FXT1\*



## ◆ Плюсы

- Большая степень компрессии чем у S3TC при компрессии 32-битовых изображений (8 против 6)

## ◆ Минусы

- На порядок большее время компрессии
- Не приемлемое качества, особенно для градиентных участков
- Не поддерживается большинством производителей

\*)3dfx Iterative Inc.

# Компрессия текстур: Формат CD\*



## Идея:

- Использование зависимости между блоками
- 2-х битовая индексная плоскость, но в блоке хранится только 1 цвет, а три других берутся из соседних 3 трех блоков

## ◆ Плюсы

- 8 кратная компрессия против 6 у S3TC

## ◆ Минусы:

- Более одного обращения в память
- Использование только 16-битного цвета

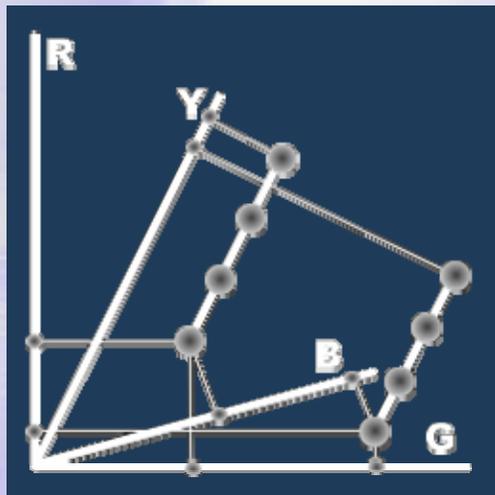
\*) CGG (Computer Graphics Group)

# Компрессия текстур: Форматы STF-8\*, STF-12\*

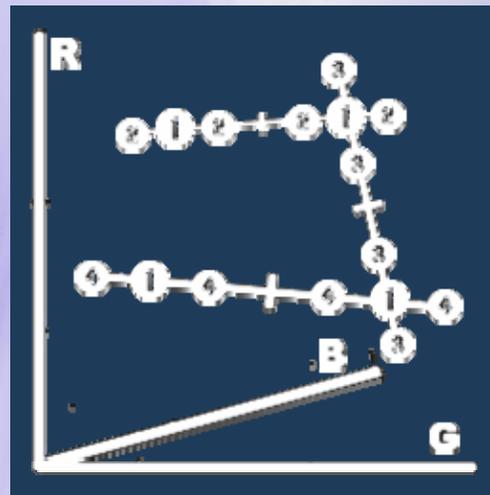


## Идеи:

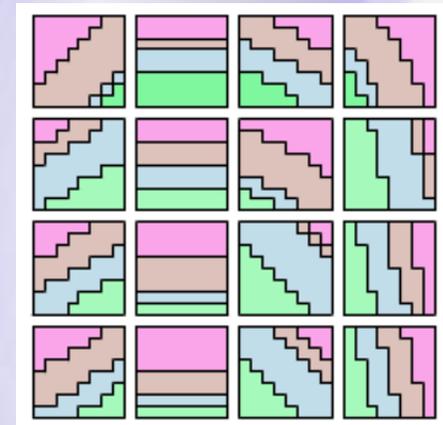
- Улучшение геометрии интерполяции палитры
- Адаптивное подразбиение блока 8x8 на 4 кластера



Палитра для STF-12



Палитра для STF-8



Примеры  
разбиений

\*) MSU Graphics&Media Lab

# Компрессия текстур: Форматы STF-8, STF-12



## ◆ Плюсы (vs S3TC):

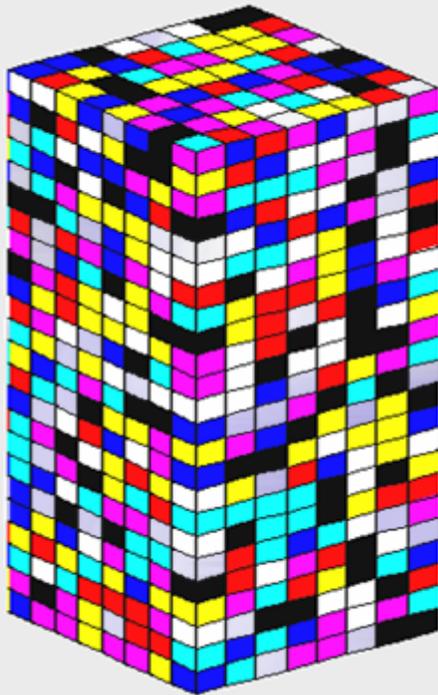
- STF-8 лучше по качеству (в среднем на 1-2 дБ) и имеет более высокую степень компрессии (8 vs. 6)
- STF-12 в 2 раза больше степень сжатия при приемлемом визуальном качестве

## ◆ Минусы

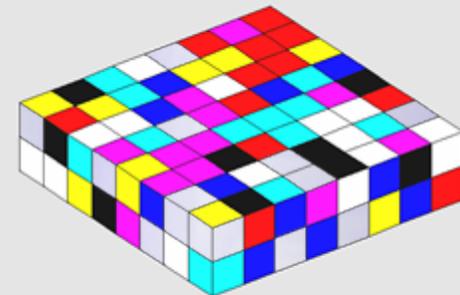
- Медленный алгоритм компрессии (более чем на 2 порядка медленнее S3TC)
- Нет поддержки прозрачности

# Исходный метод: простой и качественный

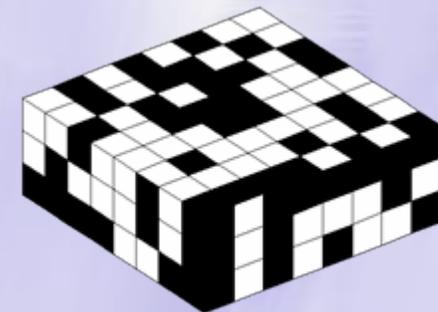
8 цветов на блок 8x8, сжатие в 4.8 раза при достаточном качестве



Палитра:



Индексы:



# Увеличение сжатие при сопоставимом качестве



## Базовые идеи:

### ◆ Сжатие индексов палитры (с потерями):

Разбиение блока на 4 кластера, и использование меньших палитр для каждого из них

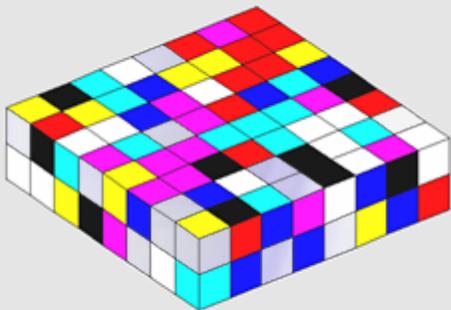
### ◆ Сжатие цветов палитры (с потерями):

Хранение только базовых цветов и аппроксимация остальных

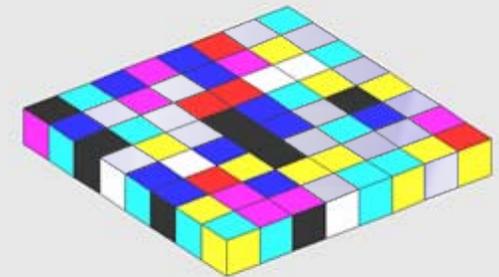
# Метод STF-8: 8-кратное сжатие



4 кластера, 8/16 цветов в палитре, 4 цвета на текстель



2-х кратное сжатие  
цветов палитры



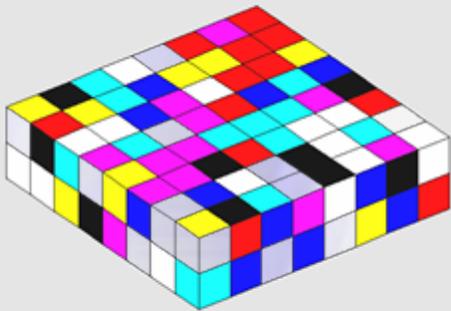
1.5/2 кратное  
сжатие индексов  
палитры



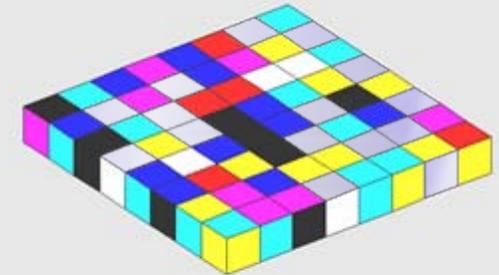
# Метод STF-8: 12-кратное сжатие



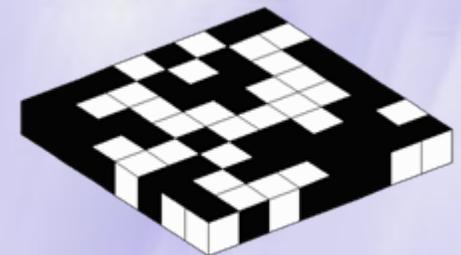
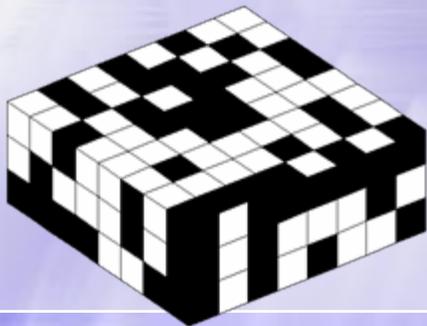
**4 кластера, 8 цветов в  
палитре, 2 цвета на текстель**



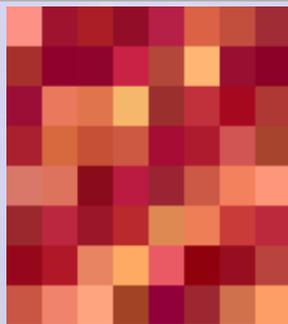
**2-х кратное сжатие  
цветов палитры**



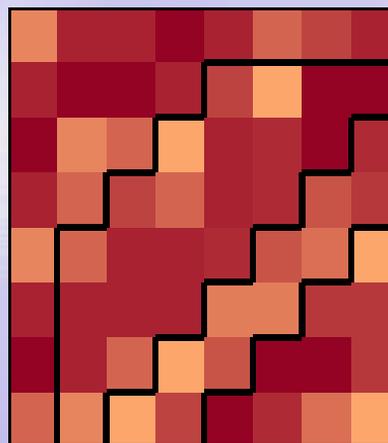
**3 кратное сжатие  
индексов палитры**



# Палитризация и кластеризация



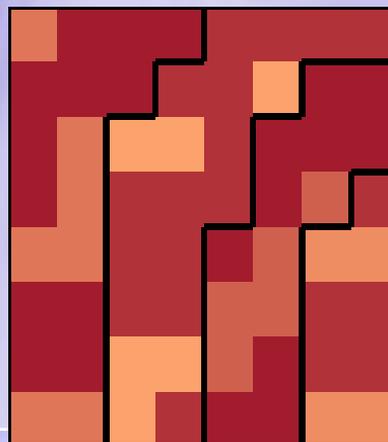
Source block



**CTF8:**  
16 color palette  
4 color in cluster

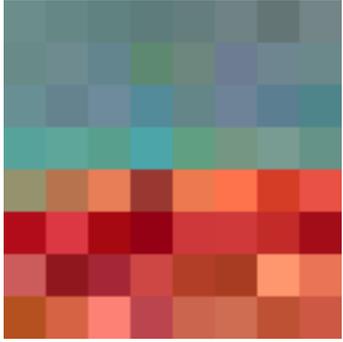
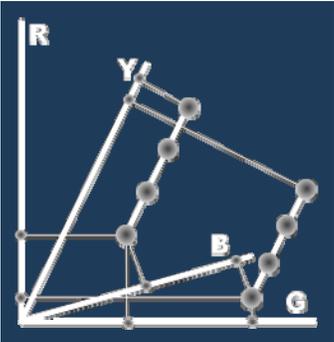
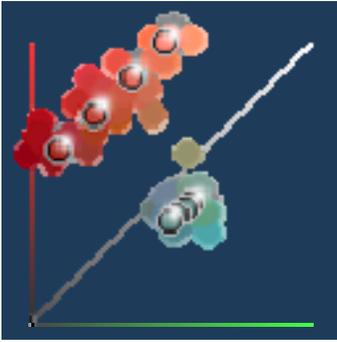
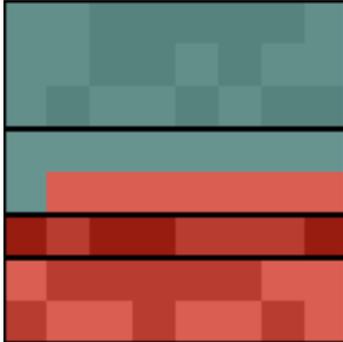
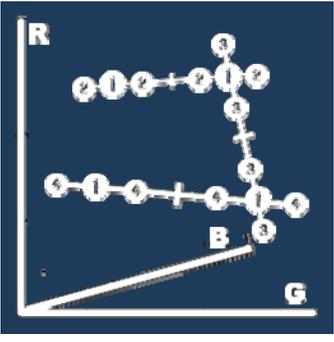
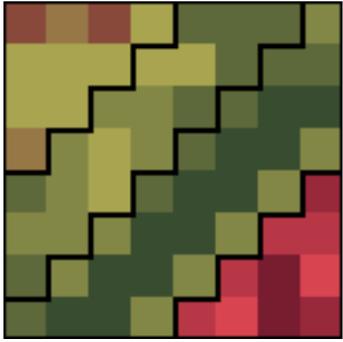


Local palette



**CTF12:**  
8 color palette  
2 color in cluster

# Компрессия текстур: Форматы STF-8, STF-12

	Исходный блок	Геометрия палитр и разбиение на кластеры		
STF-12:				
STF-8:				

# СТФ-12: Два типа палитр



**В палитре хранятся 2 базовых цвета  $C1$  и  $C2$ , в формате  $RGB-453$ , в зависимости от " $C1 < C2$ ":**

- ◆ Используется в однородных блока и во всех не цветных блоках
- ◆ Аппроксимация палитры – отрезок в пространстве между двумя базовым и цветами
- ◆ Хранятся уточняющие биты до формата  $RGB-565$

- ◆ Используется в блоках с резкими границами и в блоках с существенно разными хроматическими компонентами
- ◆ Аппроксимация палитры - 2 параллельных отрезка
- ◆ Цветовой формат отрезка –  $RGBY-4534$

# Алгоритм декомпрессии для STF-12



Компрессированный блок 128 bit

32b Палитра

8b Номер  
подразбиения  
блока

Plane[8][8]x1b  
Битовая  
плоскость

Декодирование  
8-цветовой палитры

Библиотека  
разбиений  
PartLib[256][8][8]x2b

Ind[4][2] x 3b  
Локальная  
палитра

Pal[8]x24b  
Полно-цветная палитра

PartDsp Выбранная  
палитра  
[8][8]x2b

Генерация текстелей на основе вложенной индексации

$$\text{Bitmap}[x][y] = \text{Pal}[\text{Ind}[\text{PartDsp}[x][y]] [\text{Plane}[x][y]]]$$

Декомпрессированный блок 8x8x24b = 1536 bit

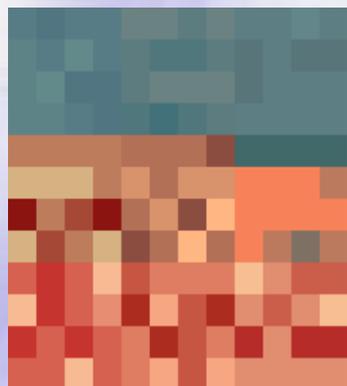
# Блочный эффект и цветовое распределение



**Исходный  
блок  
(12x12)**



**CTF-8  
(блоки 8x8)**



**S3TC  
(блоки 4x4)**

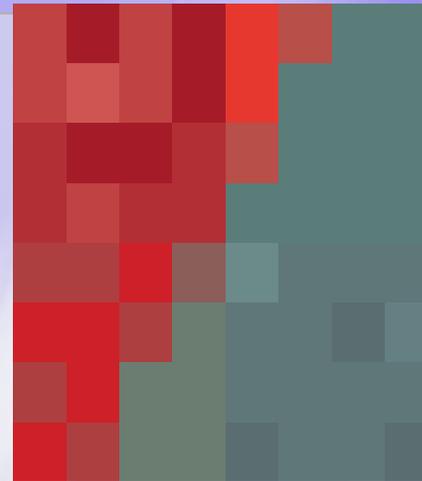


**CTF-12  
(блоки  
8x8)**

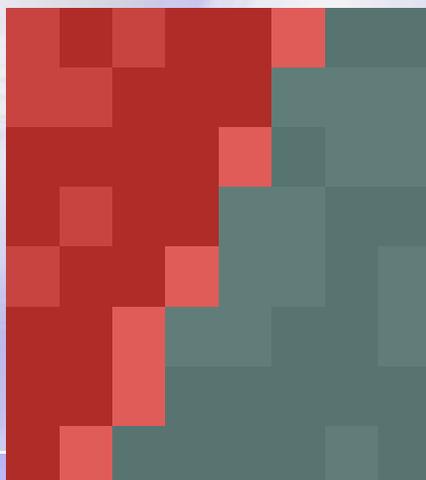
# Граничный эффект



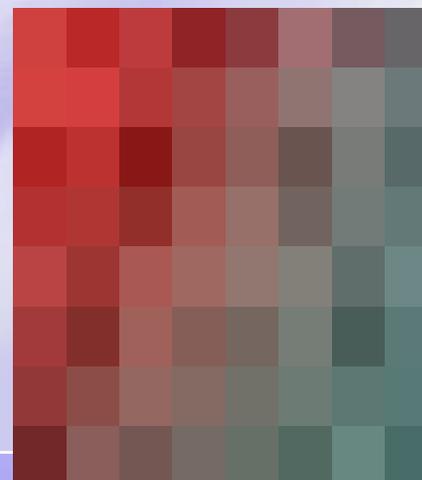
**Исходный**



**S3TC  
(6раз)**



**STF-12  
(12раз)**



**JPEG  
(12раз)**

# Сравнение S3TC с CTFs по объективным метрикам



	Параметры					LUV-метрика на 'Portrait'					
	Степень компрессии	Степень компрессии палитры	Степень компрессии и индексов	Количество цветов на блок	Размер палитры	PSNR	BLACK	GREEN	RED (%)	MAXLUV	MAXRGB
CTF-8	8	10/6	2/1.5	16/8	48b	34.0	24	42	34	48	50
CTF-12	12	6	2	8	32b	28.5	17	41	41	49	98
S3TC	6	3	1	4	32b	32.96	21	46	33	54	52

# Тестовое изображение



**Source**

**S3TC**

**CTF-12**

# Эффективность работы кэша при реальном алгоритме рендеринга



метод S3TC



метод FXT1



метод CTF-12

вариант работы	параметр	24-битное изображение	метод S3TC	метод FXT1	метод CD	метод CTF-12
1	cash hit rate	75%	87%	90.7%	78.3%	91.1%
	average latency	26.5	14.744	11.087	23.256	10.714
2	cash hit rate	75%	96.6%	97.9%	90.7%	98.7%
	average latency	26.5	5.362	4.050	11.133	3.264

Сжатые текстуры на шаре

Хранение всех МИП-МЭПОВ

Генерация МИП-МЭПОВ 'на лету'

# **СЖАТИЕ ТЕКСТУР: Генерация текстур**

*Наиболее компактный метод  
представления текстур –  
их генерация*

# Типы генерации текстур

## Процедурные текстуры:

- Алгоритмическая генерация текстур
- Для каждой физической модели свой алгоритм

## Генерация мип-мэпов:

- Универсальный алгоритм, не зависит от типа текстуры
- Дополняет алгоритм компрессии текстур
  - ◆ **Проблема:** памяти акселератора всегда мало, даже если компрессировать текстуры
  - ◆ **Выход:** не хранить, а генерировать самые детализированные мип-мэпы уровни

# Генерация мип-мэпов

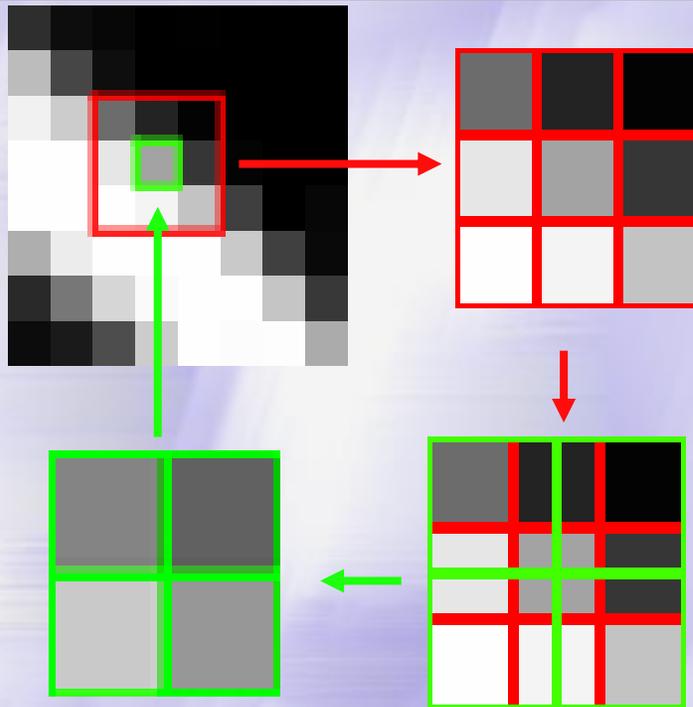
## ◆ Требования:

- Реалистичность в не зависимости от типа и разрешения текстуры
- Высокая скорость и возможность аппаратной реализации

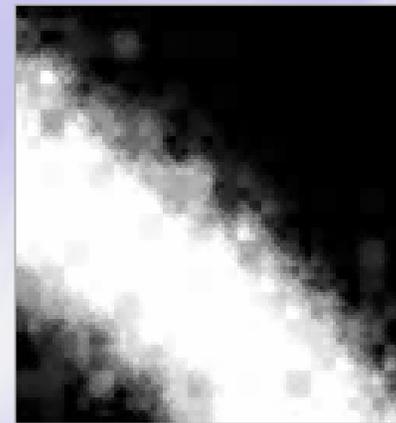
## ◆ Подход: вероятностная генерация

- Метод№1: фрактально-каскадная генерация с вероятностно-распределенным локальным коэффициентом подобия масштабных уровней
- Метод№2: генерация с вероятностным законом положения и расположения шаблонов

# Фрактально-каскадный метод генерации



Рекурсивное  
фрактально-каскадное  
подразбиение

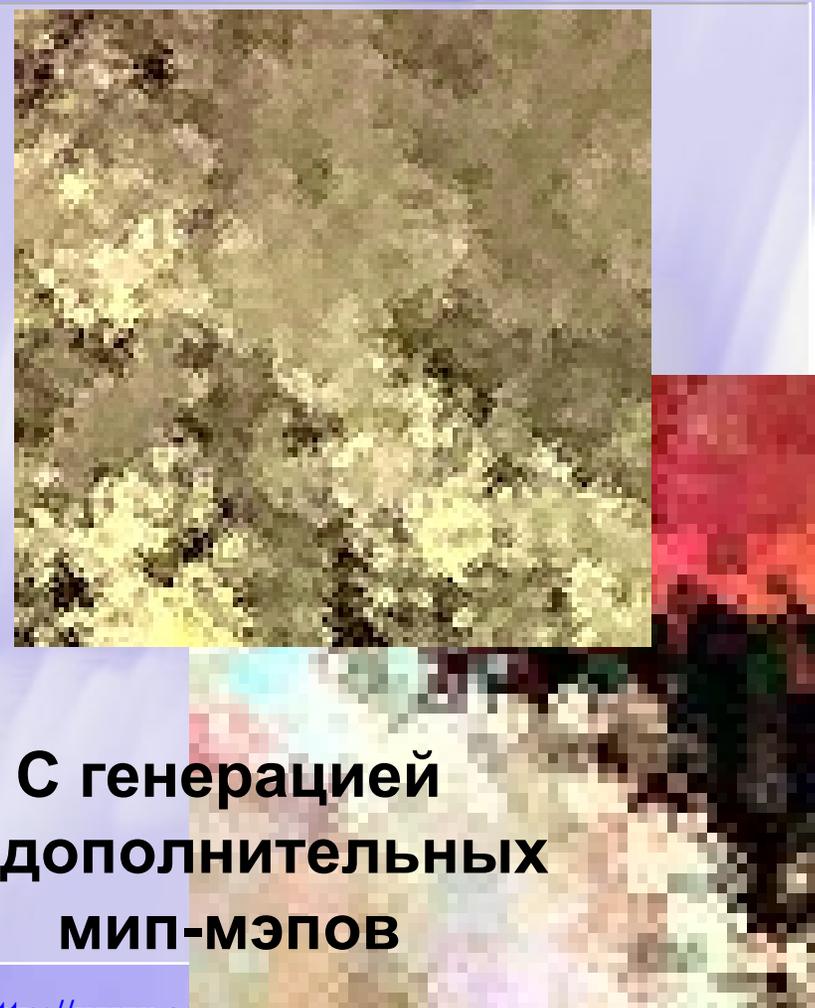
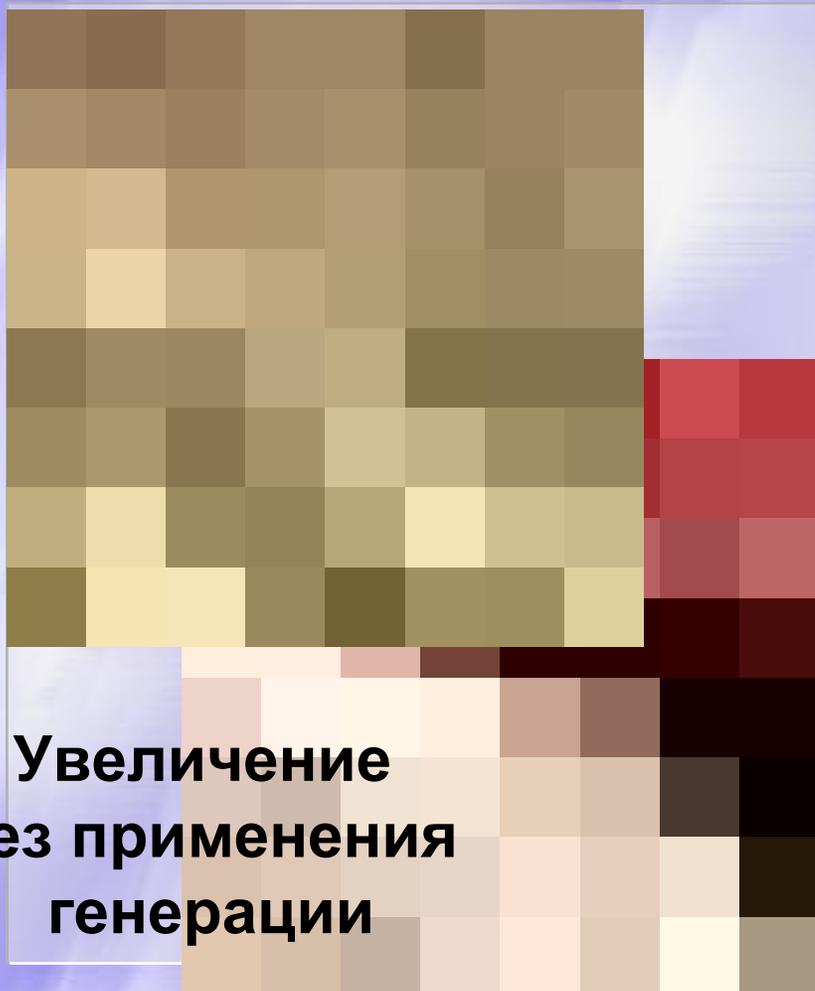


После 8 итераций

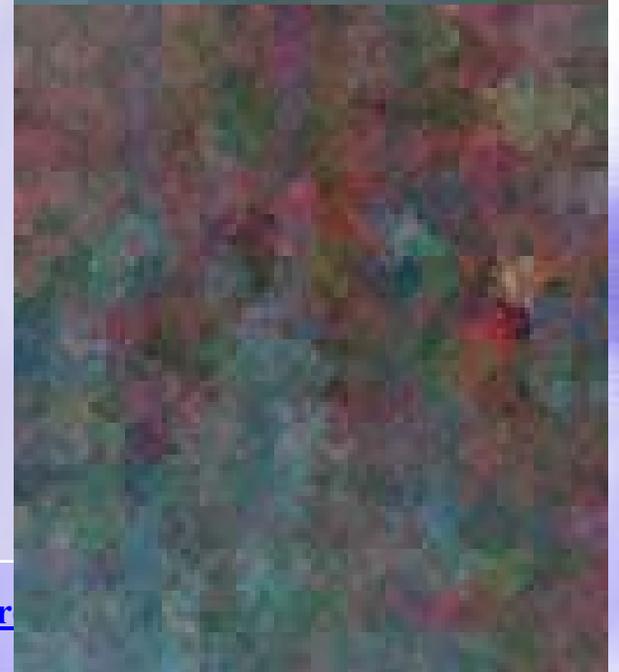
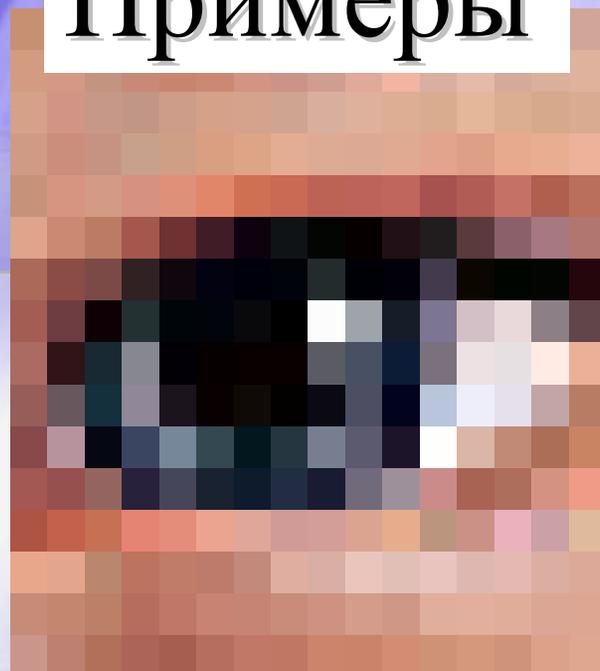
$$A = \alpha B + \beta,$$

$\alpha = N(0, \sigma), \beta = N(0, \sigma')$

# Фрактально-каскадный метод генерации



# Примеры



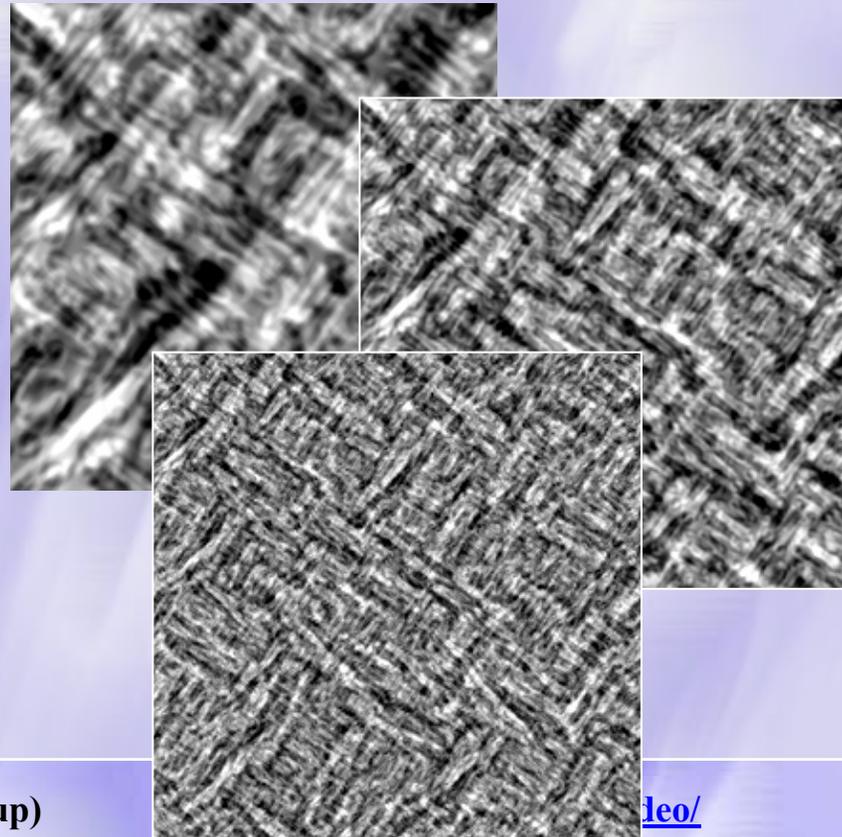
# Многомасштабная генерация с использованием шаблонов



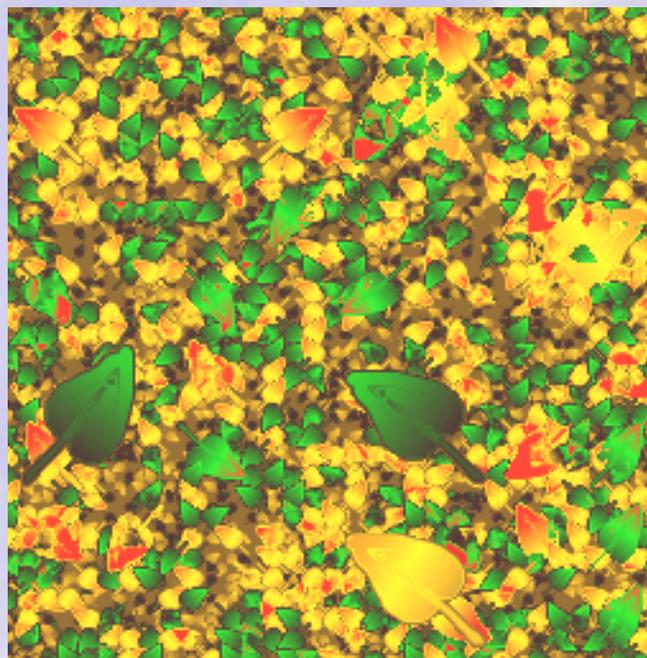
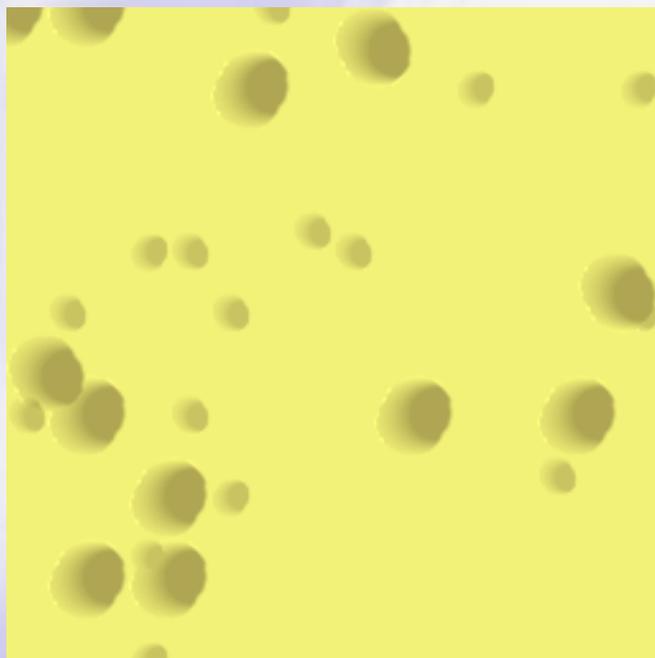
## Многомасштабные Шаблоны



## Различные уровни детализации сгенерированных текстур



# Многомасштабная генерация с использованием шаблонов



$4n$  операций/текстель ( $n$  – количество масштабных уровней)

# Задания



- ◆ Задания по курсу расположены на странице курса:

<http://graphics.cs.msu.su/courses/mdc/>

- ◆ Следующая тема - сжатие аудиоданных