3D графика и трасиране на лъчи v.6.0

http://raytracing-bg.net/

Тема 8

Триъгълни мрежи (продължение)
Файлови формати за триъгълни мрежи
Bump Maps

Съдържание

- Пресичане на триъгълни мрежи
- Файлови формати за триъгълни мрежи
- Форматът .OBJ
- Релефни текстури (Bump maps)

Представяне на триъгълната мрежа

- Триъгълната мрежа е просто списък с триъгълници
  - Можем да ги пазим в един vector<>, като за всеки триъгълник помним 3-те му върха и нормалата му
  - Но това е неефективно: в един реалистичен mesh, доста от върховете съвпадат (споделени са между няколко триъгълника)
    - Същото се отнася и за нормалите на върховете, често и за текстурните (UV) координати
  - Ето защо ще пазим един списък с всички върхове, нормали и UV координати, а триъгълниците ще индексират в тези масиви

В код (revisited)

```
struct Triangle {
   int v[3], n[3], t[3];
   Vector gnormal; // geometric normal of the triangle itself
};

class Mesh: public Geometry {
   vector<Vector> vertices;
   vector<Vector> normals;
   vector<Point> uvs;
   vector<Triangle> triangles;
...
```

Пресичане с триъгълна мрежа

- Алгоритъм за пресичане с триъгълна мрежа:

```
procedure intersectMesh(ray, info):
    minDist = +∞
    foreach triangle T in mesh:
        distance = intersectTriangle(T, ray, tempInfo)
        if distance < minDist:
            minDist = distance
            info = tempInfo
    return minDist
```

Пресичане с триъгълник

- Дадено:
  - A, B, C – върхове на триъгълника
  - O – начало на лъча
  - D – посока на лъча
- Търси се:
  - λ_2, λ_3
  - γ – разстоянието до пресечната точка

Пресичане с триъгълник

- Пресечната точка може да дефинираме така:

  $X = O + \gamma D = A + \lambda_2(B-A) + \lambda_3(C-A)$

  Т.е.,

  $\lambda_2(B-A) + \lambda_3(C-A) - \gamma D = O - A$

- Това е система линейни уравнения (3 уравнения, 3 неизвестни):

  $\lambda_2(B.x-A.x) + \lambda_3(C.x-A.x) - \gamma D.x = O.x - A.x$

  $\lambda_2(B.y-A.y) + \lambda_3(C.y-A.y) - \gamma D.y = O.y - A.y$

  $\lambda_2(B.z-A.z) + \lambda_3(C.z-A.z) - \gamma D.z = O.z - A.z$

Пресичане с триъгълник

- Можем да приложим правилото на Крамер за намиране решението на такава линейна система
- Забележка: може да пресмятаме детерминантите на матриците по следния начин: тъй като всеки стълб от матрицата е 3D вектор, то самата детерминанта е смесеното произведение от тези 3 вектора. Т.е. ако стълбовете са вектори a, b и c, то детерминантата се изчислява като:

  Det = (a ^ b) * c

Решение на системата

- $Det_{cr} = ((B-A)^{(C-A)}) * -D$
- Нека $H = (O-A)$

  $\lambda_2 = ((H^{(C-A)}) * -D) / Det_{cr}$

  $\lambda_3 = (((B-A)^H) * -D) / Det_{cr}$

  $\gamma = (((B-A)^{(C-A)}) * H) / Det_{cr}$

Файлови формати за триъгълни мрежи

- Удобно е триъгълните мрежи да се записват във файл
- Поради големината си, повечето формати за триъгълни мрежи са двоични
  - Например форматът на 3ds max - .3DS
  - Но има и изключения
- Във файловете с триъгълни мрежи обикновено се записва само геометрията – няма материали
  - Записват се списъци с върхове, нормали и текстурни координати, както и списъка със самите триъгълници

Файловият формат .OBJ

- Разработен първоначално от Wavefront Technologies, но е универсално приет в повечето 3D пакети
  - 3ds max, Maya, XSI, Blender, дори Mathematica, както и много други
  - Прост текстов формат, с добра [спецификация]
- След като си напишем .obj четец, ще можем да ползваме триъгълни мрежи от почти всички 3D пакети (почти всички от тях имат .obj exporter)

OBJ пример – тетраедър

```
# XSI Wavefront OBJ Export v3.0

#begin 5 vertices
v 0.000000 -1.333333 0.000000
v 0.000000 2.666667 0.000000
v -1.000000 -1.333333 -0.000000
v 0.500000 -1.333333 0.866025
v 0.500000 -1.333333 -0.866025
#end 5 vertices

#begin 6 normals
vn 0.000000 -1.000000 0.000000
vn -0.496139 0.124035 0.859338
vn 0.000000 -1.000000 0.000000
vn 0.992278 0.124035 0.000000
vn 0.000000 -1.000000 0.000000
vn -0.496139 0.124035 -0.859338
#end 6 vertex normals

#begin 6 faces
f    4//1    3//1    1//1
f    3//2    4//2    2//2
f    5//3    4//3    1//3
f    4//4    5//4    2//4
f    3//5    5//5    1//5
f    5//6    3//6    2//6
#end 6 faces
```

OBJ

- „v *X Y Z*“ описва един връх
- „vt *U V*“ описва двойка текстурни координати
- „vn *X Y Z*“ описва нормален вектор
- „f $връх_1$ $връх_2$ ... $връх_N$“ описва един многоъгълник
  - $връх_i$ е или число, или тройка *връх/tex_uv/нормала*
  - Числата индексират в списъците v, vt и vn (1-based)
  - Може да имаме повече от 3 върха – например „f 1 2 3 4“ дефинира четириъгълник

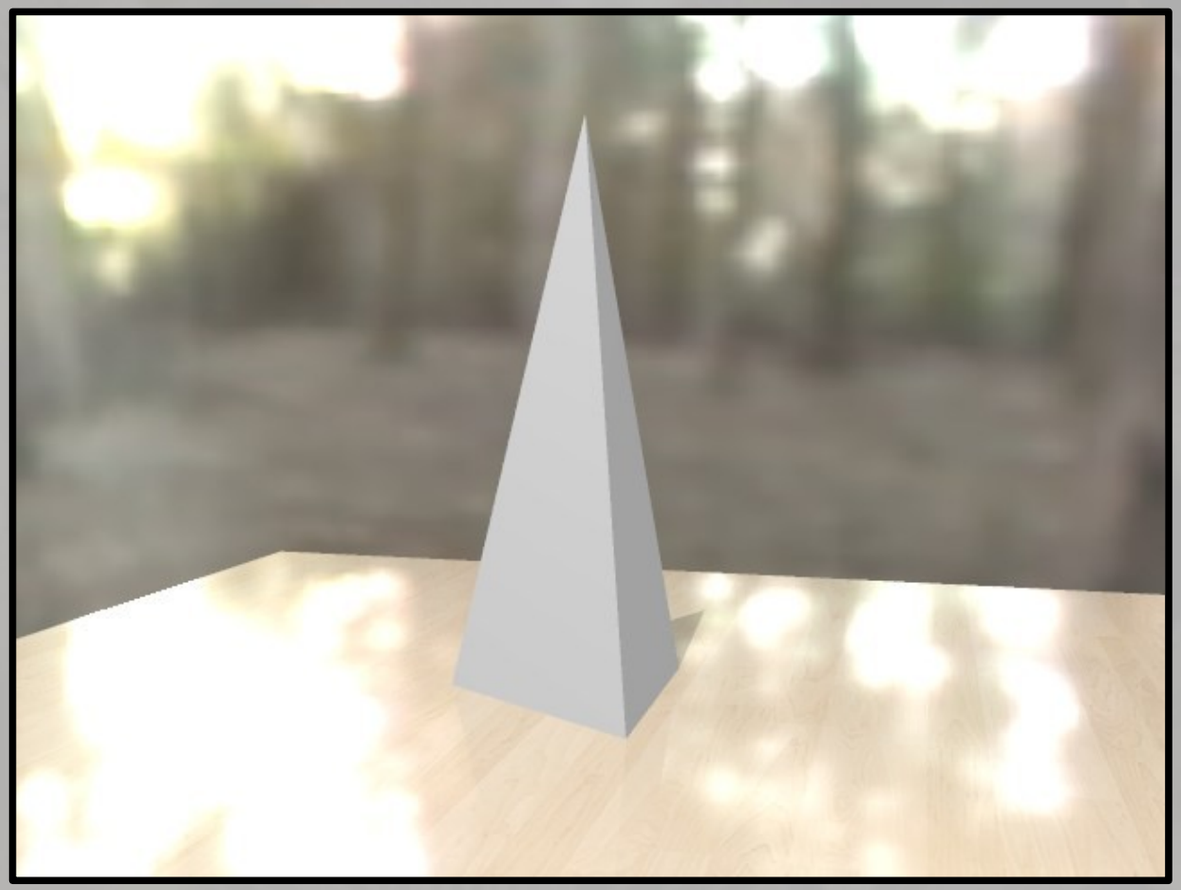
OBJ

- vt и vn списъците може да липсват. Тогава OBJ файла дефинира само формата на геометрията
- Някой от компонентите на тройката връх/tex_uv/нормала може да липсва
  - Например, „5//1“ или „5/4“
- При многоъгълник с повече от 3 върха, гарантирано е, че те лежат на една равнина

Схема за OBJ четец

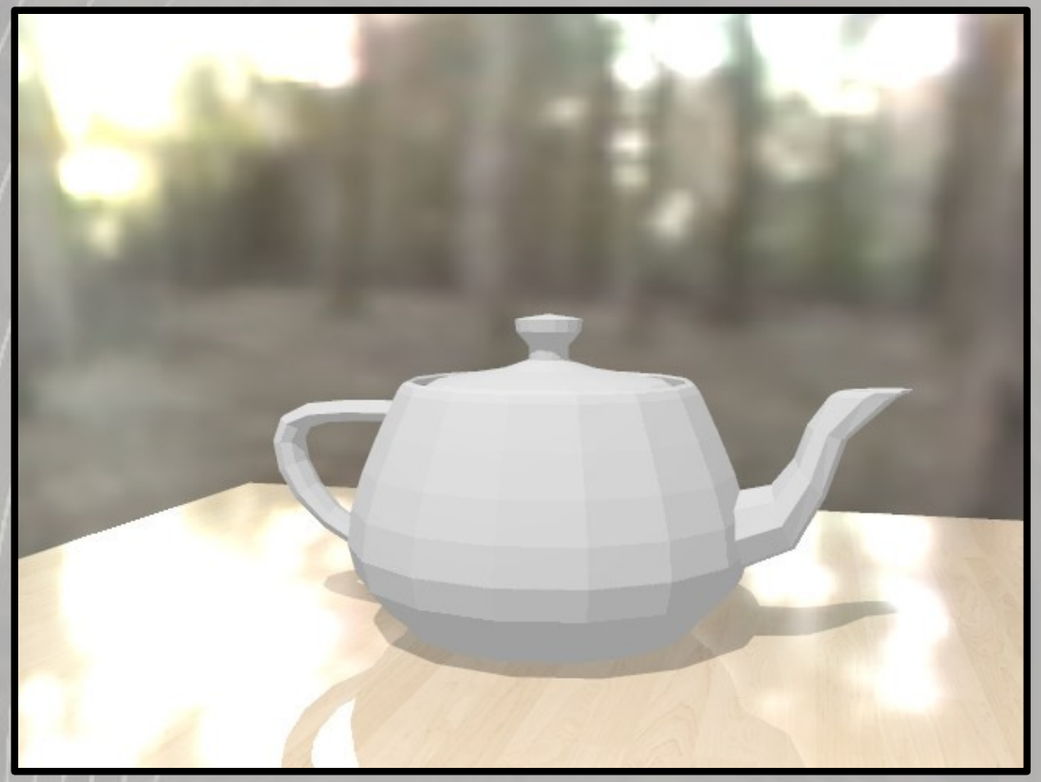
- Прочитат се всички v, vt, vn и f редове и се вкарват в съответните масиви
- Преизчисляват се нормалите (gnormals) на триъгълниците
- Като оптимизация - намира се заграждащата сфера на обекта
  - Или друга заграждаща геометрия (bounding volume)

Резултати



Тетраедър

Резултати



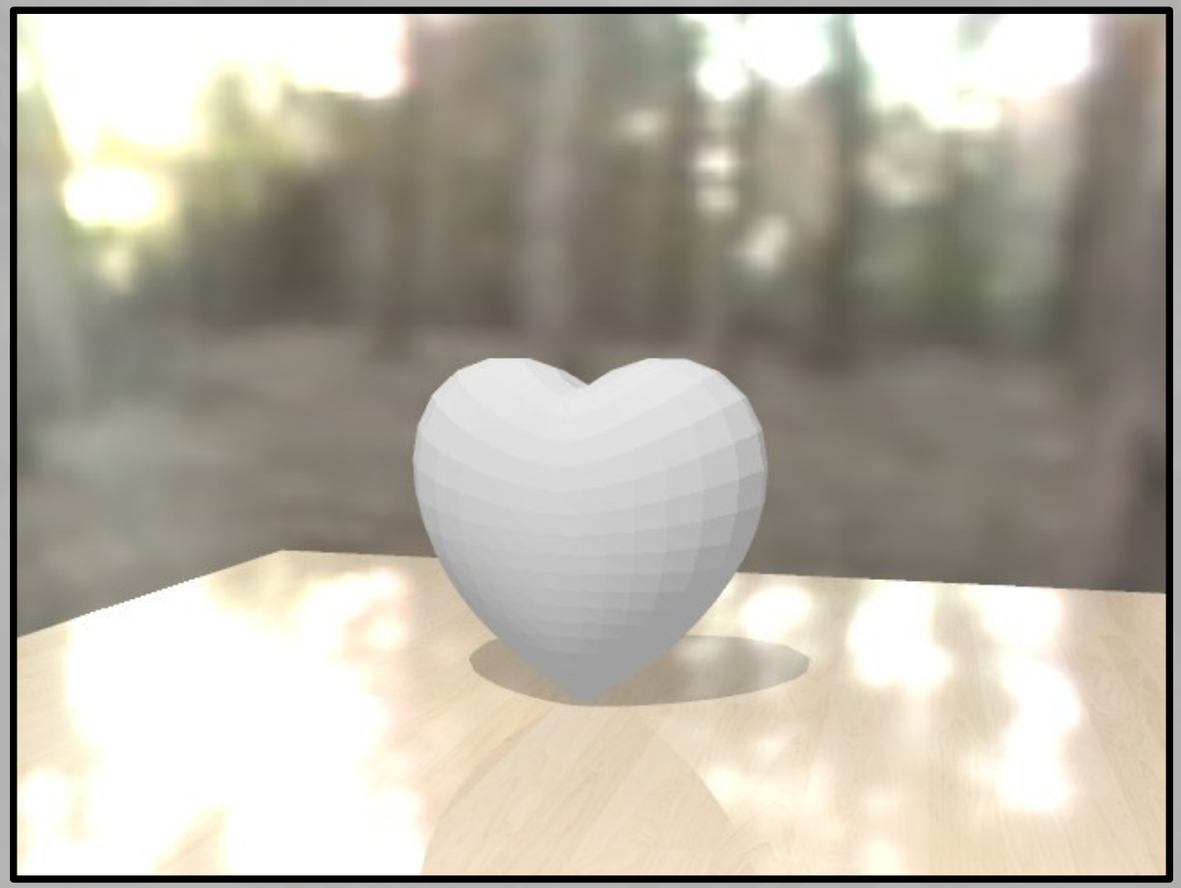
Чайник

- Без и с изглаждане на нормалите

Текстури

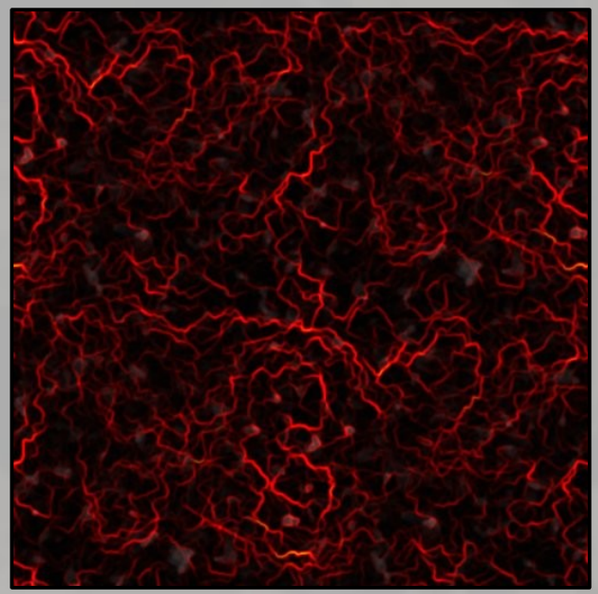
- Чайникът си има текстурни координати

Резултати



Сърце (с lambert шейдър)

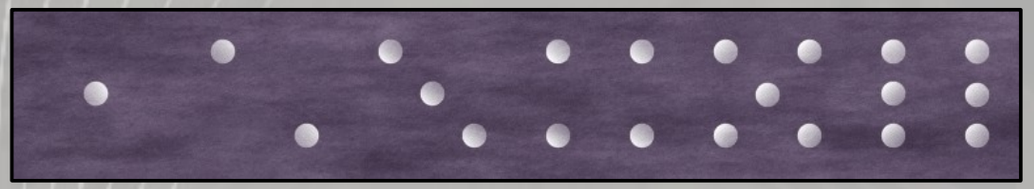
Резултати

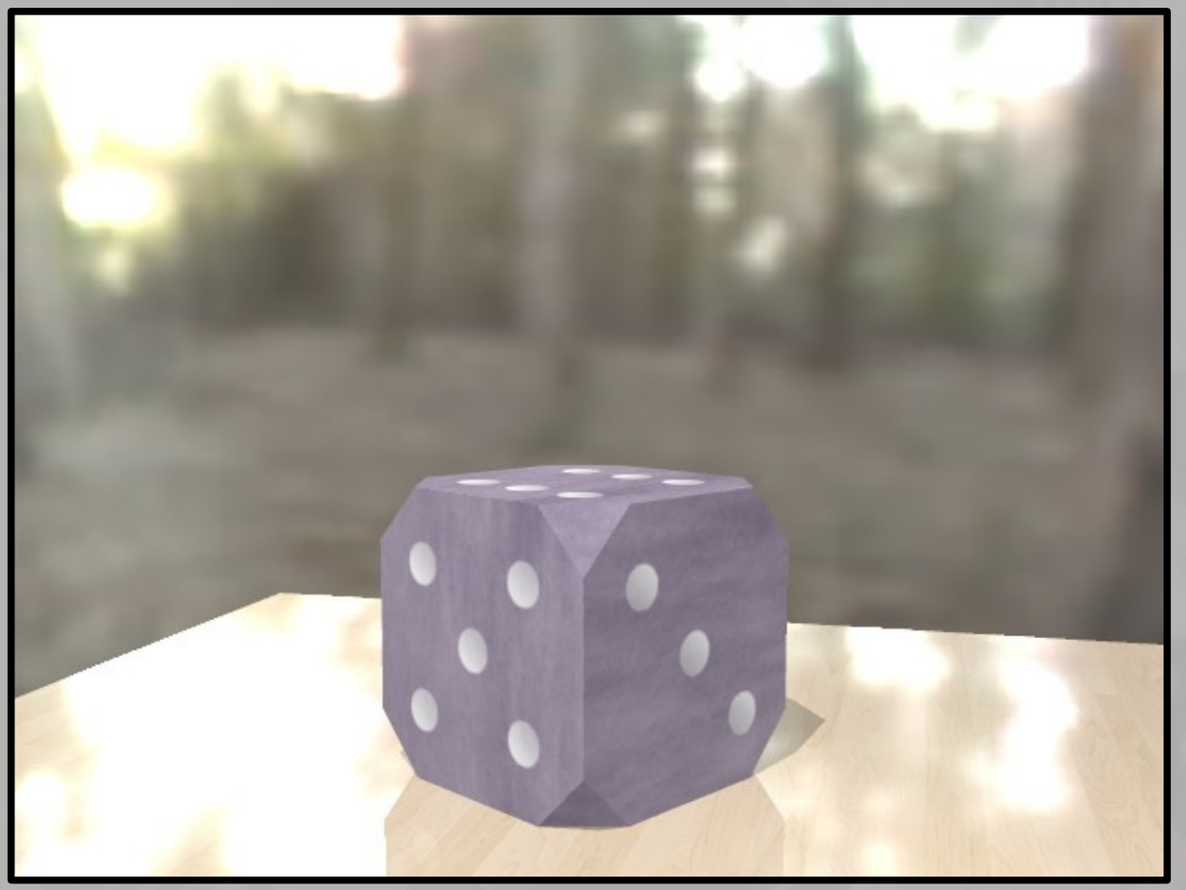


С текстура

Резултати

- Текстурата не е необходимо да е квадратна
- Текстурните координати на съседните върхове може да са прекъснати



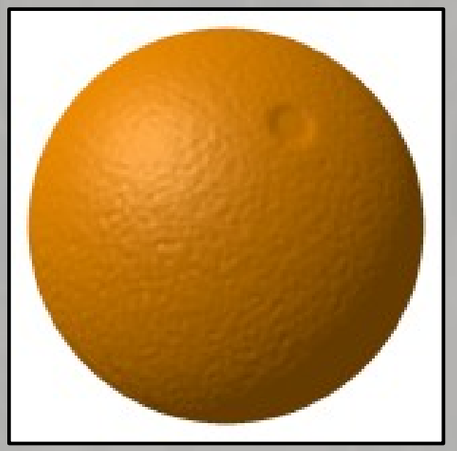


Резултати

Сложен mesh с fresnel шейдър, наподобяващ стъкло

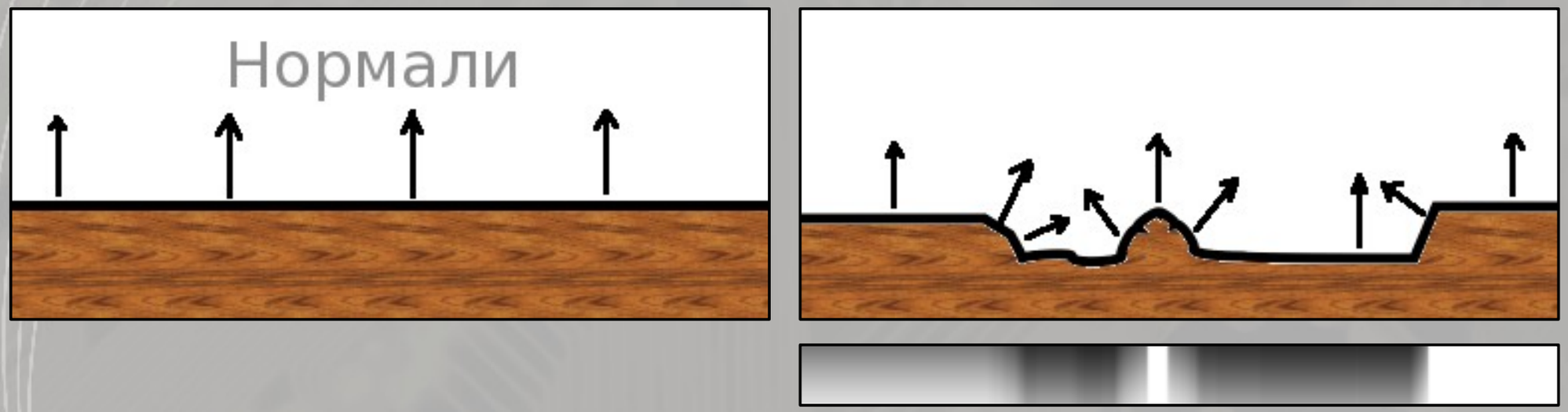
Bump mapping

- Bump map-овете симулират релеф върху повърхностите (които иначе са гладки)
- Подходящи са когато релефът е относително дребнозърнест
- Позволяват ни да добавим усещане за фин детайл, и то почти „безплатно“ (не генерираме никаква допълнителна геометрия)



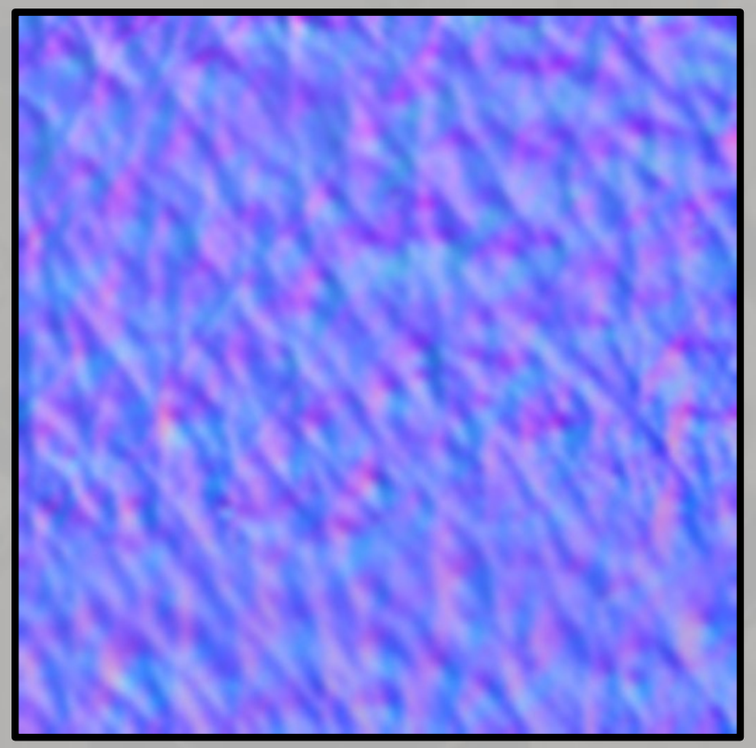
Bump mapping

- Идеята: при пресичането с геометрията, ще преместим малко нормалата. Колко и накъде точно да я преместим ще вземем от някаква текстура
- Обикновено текстурата показва „височината“ на релефа



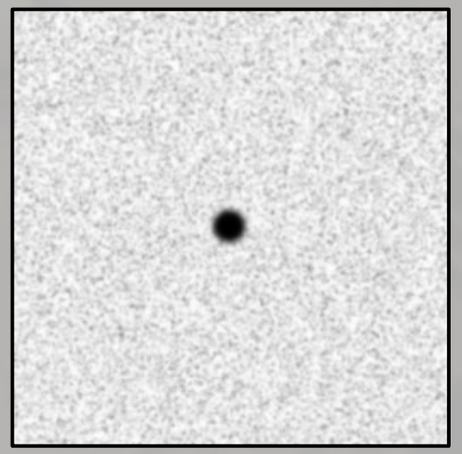
Видове bump maps

- Вариант 1: текстурата показва с по колко (по x, y, и z направленията) да отклоним нормалния вектор. Т.е., от RGB компонентите на цвета в дадения тексел прехвърляме към XYZ компоненти на отклоняващия вектор
- Пример за такава текстура:



Видове bump maps

- Вариант 2: текстурата ни показва „височината“ на релефа
- Пример:



- Този вариант изисква да изчислим една специална текстура на отклонението (подобна на тази от Вариант 1). Тя се получава като пресметнем разликите между пикселите от дадената текстура по X и по Y (т.е., като „диференцираме“ текстурата)

Изчисляване на bump map текстурата

for each row y:

    for each column x:

        bumpTex(x, y).dx = origTex(x, y).intensity() - origTex(x + 1, y).intensity()

        bumpTex(x, y).dy = origTex(x, y).intensity() - origTex(x, y + 1).intensity()

* (трябва само да решим проблема ако липсват десните/долните съседи)

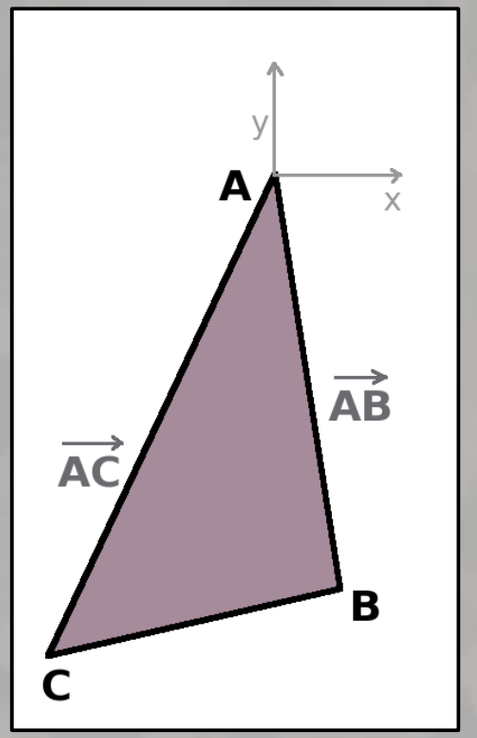
** (изваждаме *ляв-десен*, защото искаме, ако левият пиксел е по-ярък, отклонението да е положително по x (виж схемата 3 слайда назад), т.е. по-ярките пиксели да са по-високи)

Реализация на bump map

- В Mesh::intersect() трябва да променим info.normal нормалата, като вземем силата на отклонението от bump map текстурата, индексирайки я с изчислените uv координати
- Обаче не знаем накъде да отклоним нормалата
  - Двойката отклоняващи вектори dNdx и dNdy зависят от текущата позиция по триъгълната мрежа
  - Например, върху горния капак на един куб, dNdx = +x, dNdy = +z. Върху дясната страна (+X страната), те са dNdx = +z, dNdy = +y

Реализация на bump map

- Идеята: в текстурното (2D) пространство, ще изразим векторите (1, 0) и (0, 1) спрямо координатната система на триъгълника (т.е., спрямо барицентричните координати)
  - (заб: тук, A, B и C са текстурни координати; т.е. върховете на триъгълника в UV пространството)
- В резултат на това, ще имаме
  - x = px * AB + qx * AC
  - y = py * AB + qy * AC



Реализация на bump map

- След като сме сметнали px, qx, py и qy, можем да се прехвърлим в истинското 3D пространство, и, ползвайки върховете на триъгълника A, B и C, да пресметнем отклоняващите вектори в 3D:
  - dNdx = px * AB + qx * AC
  - dNdy = py * AB + qy * AC
  - (трябва само да ги нормираме)
- Така, изчисляването на bump mapping-а става:
  - info.norm += bumpTex(u, v).dx * dNdx + bumpTex(u, v).dy * dNdy

Резултати

Bump map върху други примитиви

- Така разглежданият bump map алгоритъм ще работи само върху триъгълни мрежи, заради начина на получаване на dNdx и dNdy
- Но ако реализираме коректното пресмятане на тези вектори върху друга геометрия, ще може да приложим bump map и върху нея
- Вариант 1 на bump map работи навсякъде, но се изисква специално приготовление на самия bump map, т.е. текстурата трябва да „знае“ върху каква геометрия ще се прилага

Разширения на bump map идеята

- Parallax mapping
  - При него, модифицираме и текстурните координати (симулира се къде щеше да удари лъча, ако геометрията наистина е релефна)
- Displacement mapping
  - Триъгълната мрежа се разбива до по-фини триъгълници, които действително се изместват