

3D графика и трасиране на лъчи v.4.0



<http://raytracing-bg.net/>

Тема 8

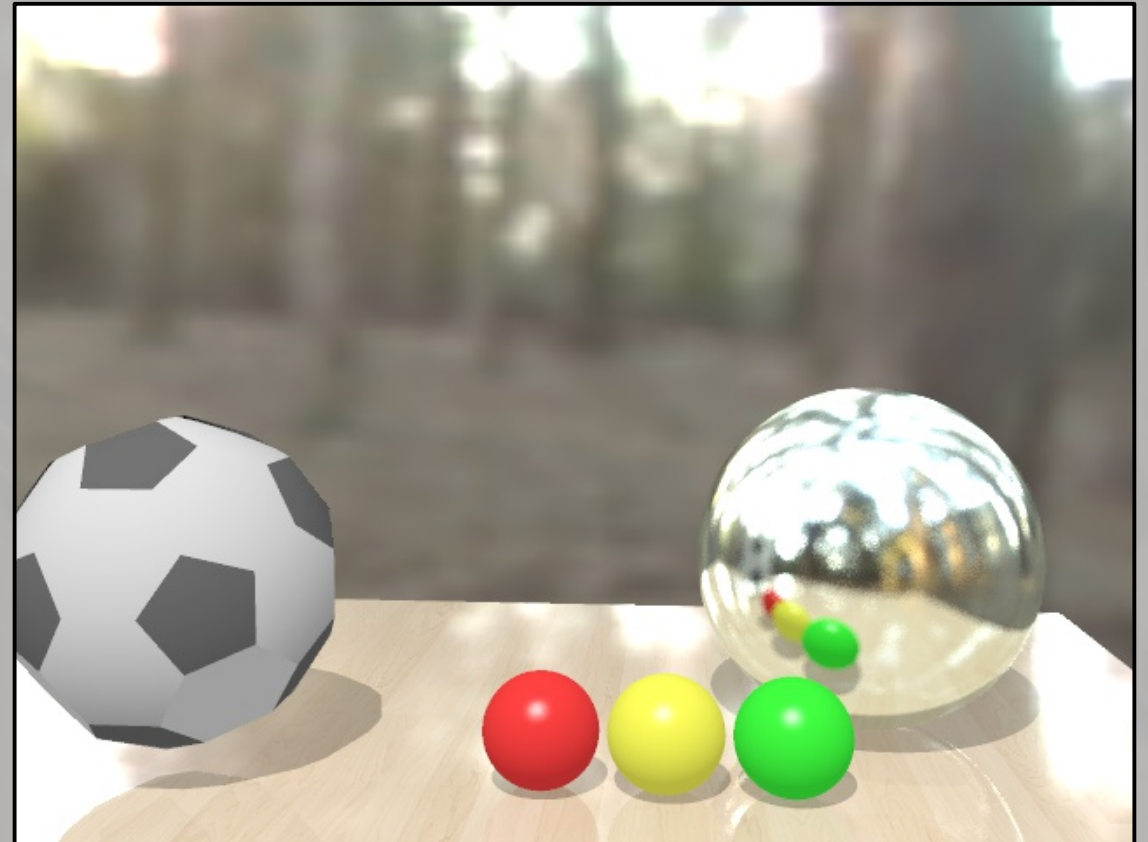
Триъгълни мрежи (продължение)
Файлови формати за триъгълни мрежи
Bump Maps

Съдържание

- Новини
- Нововъведения в учебния рейтрейсър
- Файлови формати за триъгълни мрежи
- Форматът .OBJ – реализация
- Релефни текстурни (Bump maps)

Нововъведения в учебния рейтрейсър

- Random generator
- Scene reader
 - Сцената се чете от файл, вече не се генерира от `initializeScene()`



Откъс от входния файл

```
GlobalSettings {
  frameWidth      640
  frameHeight     480
  ambientLight   (0.5, 0.5, 0.5)
  lightPos        (-90, 1200, -750)
  lightColor      (1, 1, 1)
  lightPower      1200000
}

Camera camera {
  pos      (45, 120, -300)
  aspect   1.33333
  yaw      5
  pitch    -5
  roll     0
  fov      60
}

Plane floor {
  y      -0.01
  limit  200
}
```

Откъс от входния файл (2)

```
BitmapTexture wood_tex {
    file "texture/wood.bmp"
    scaling 400
}

Lambert floor_diffuse {
    texture wood_tex
}

Refl mirror {
}

Fresnel mirror_fresnel {
    ior 1.33
}

Layered floor_shader {
    layer floor_diffuse (1, 1, 1)
    layer mirror (1, 1, 1) mirror_fresnel
}

Node floor {
    geometry floor
    shader floor_shader
}
```

Предимства

- Няма нужда да се прекомпилира qdamage за всяка дребна промяна на сцената
- Сцената е отделена от raytracer-а
 - Пакетиране: можете да пратите сцена (+прилежащите ѝ текстури) по e-mail например
 - Можем да имаме няколко различни сцени и да се прехвърляме лесно от една на друга (просто променяме кой да бъде входния файл)
- По-малко писане
- Scriptability

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 популярния формат 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
```

ОВЈ

- „v X Y Z“ описва един връх
- „vt U V“ описва двойка текстурни координати
- „vn X Y Z“ описва нормален вектор
- „f връх₁ връх₂ ... 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

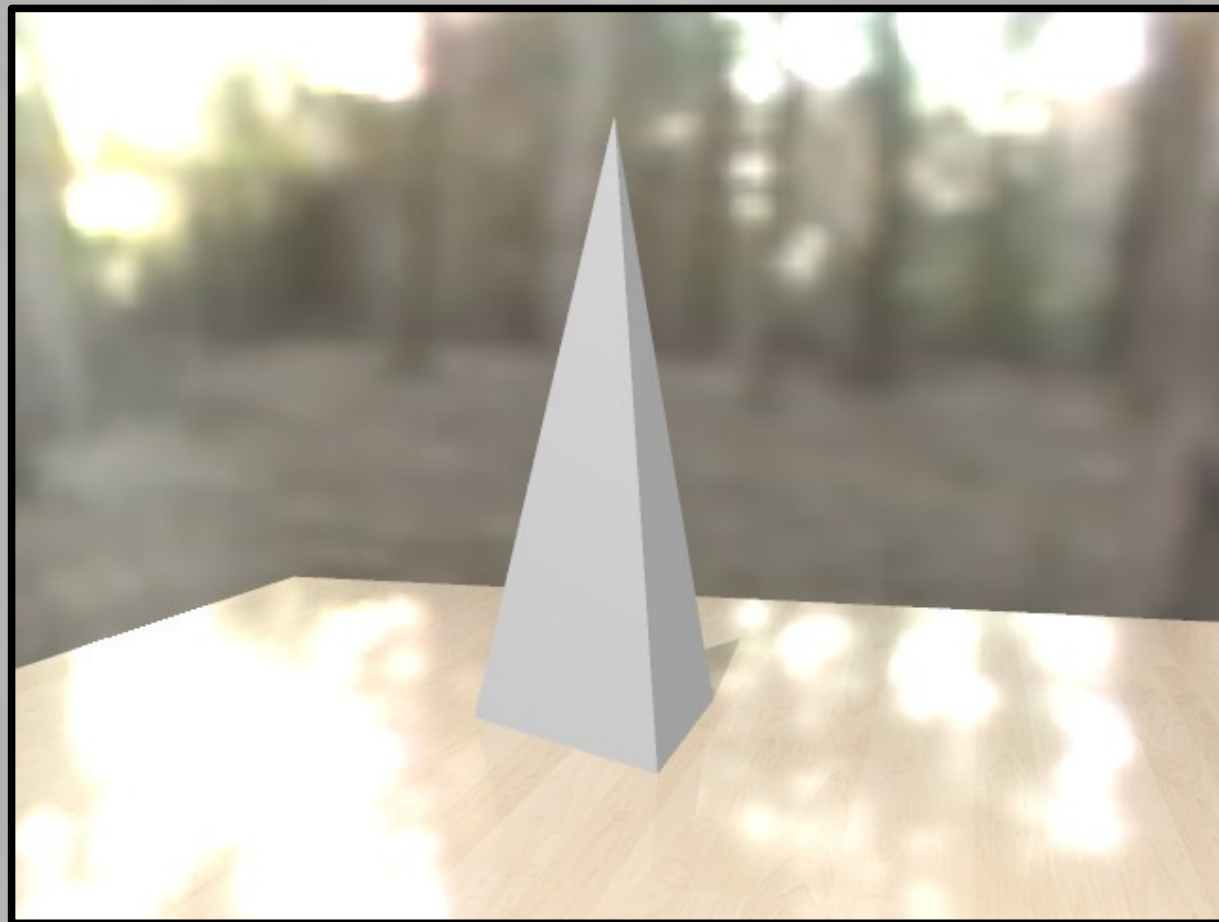
OVJ

- vt и vn списъците може да липсват. Тогава OVJ 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 ОВЈ четец

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

Резултати



Тетраедър

Резултати



Чайник

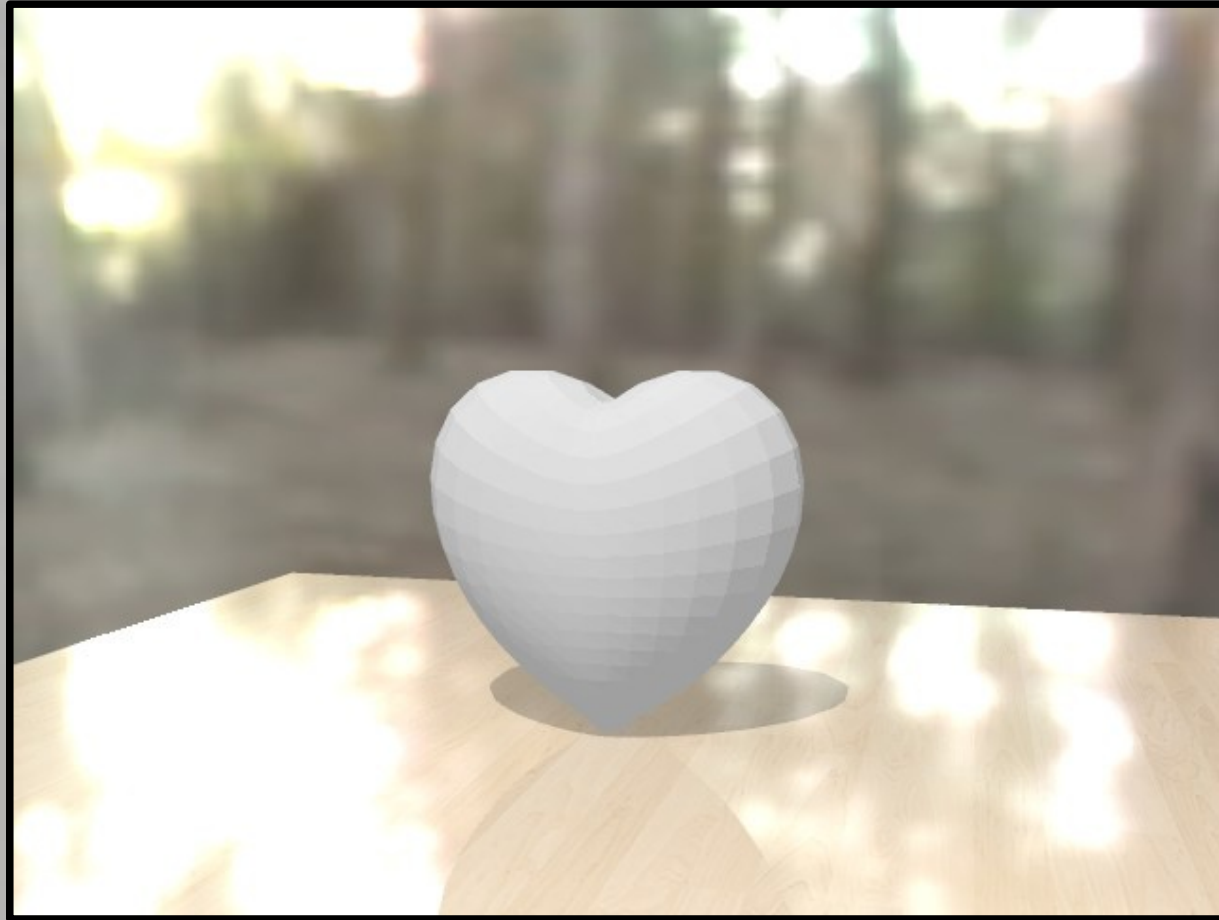
- Без и с изглаждане на нормалите

Текстури



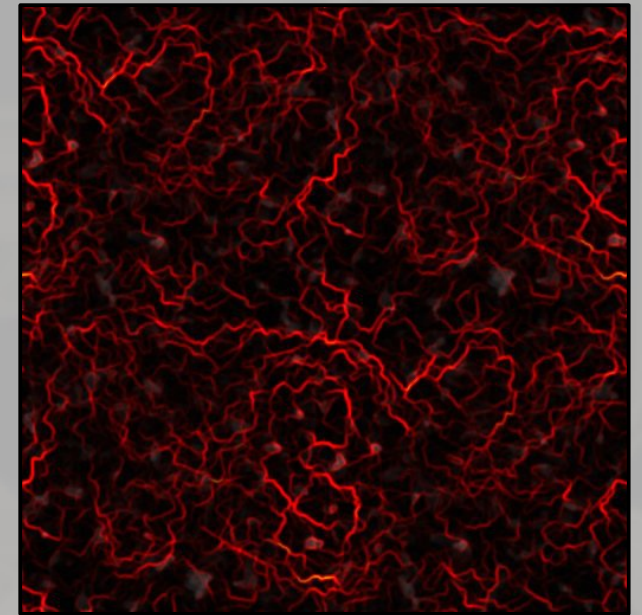
- Чайникът си има текстурни координати

Резултати



Сърце (с lambert шейдър)

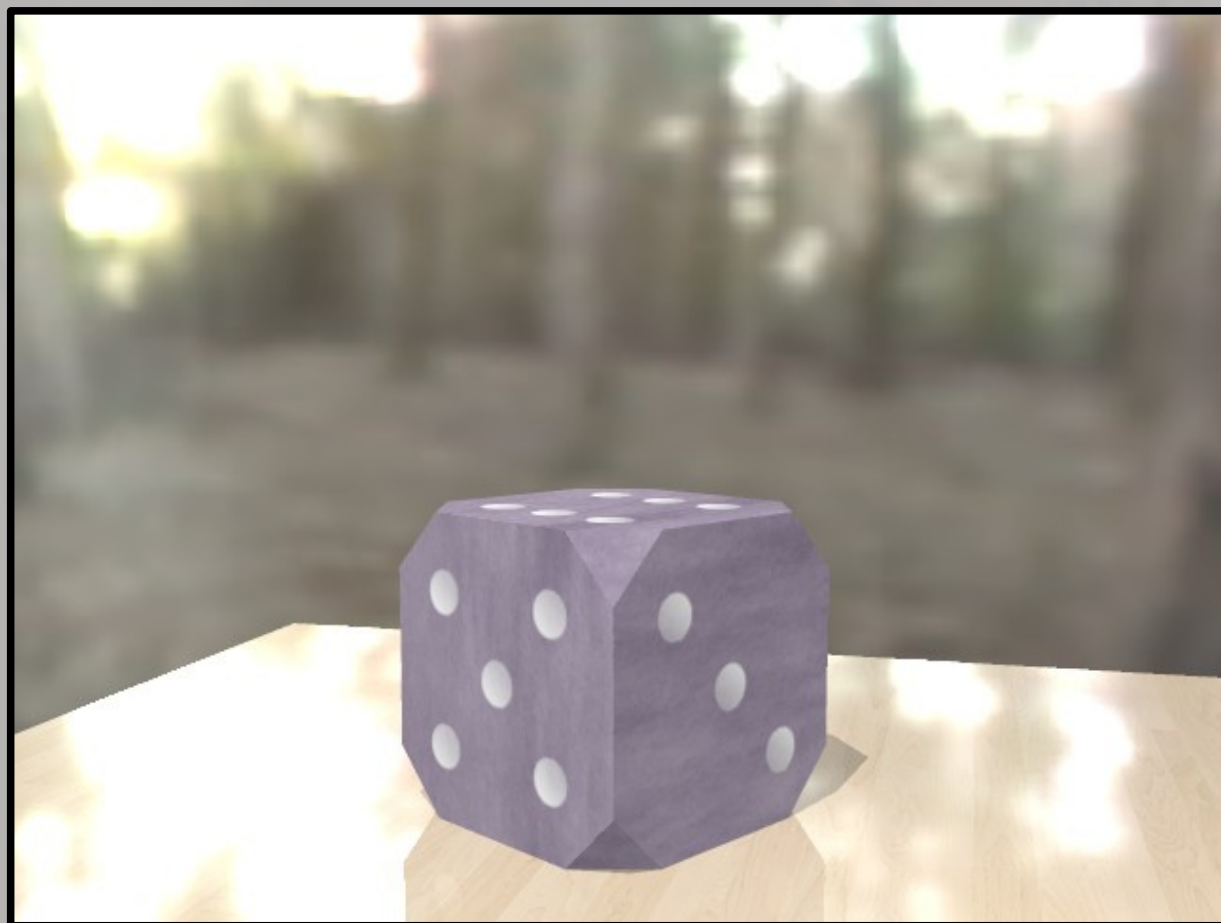
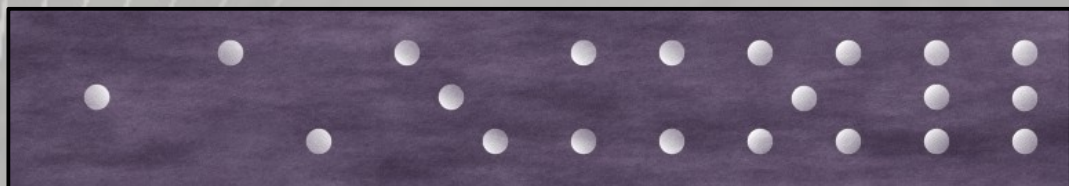
Результати



С текстура

Резултати

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



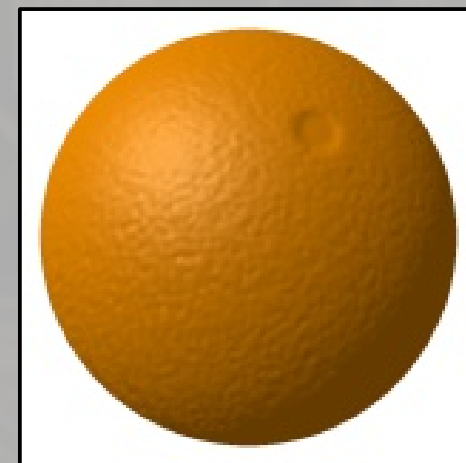
Резултати



Сложен mesh с fresnel шейдър, наподобяващ стъкло

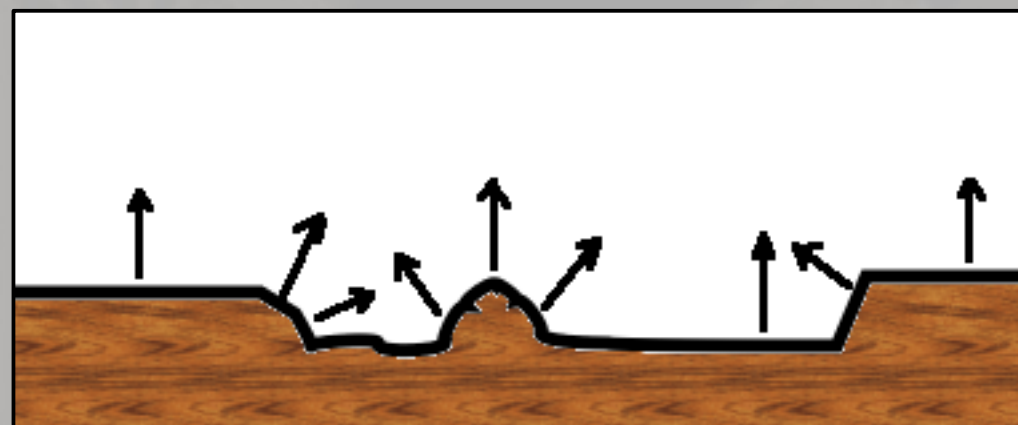
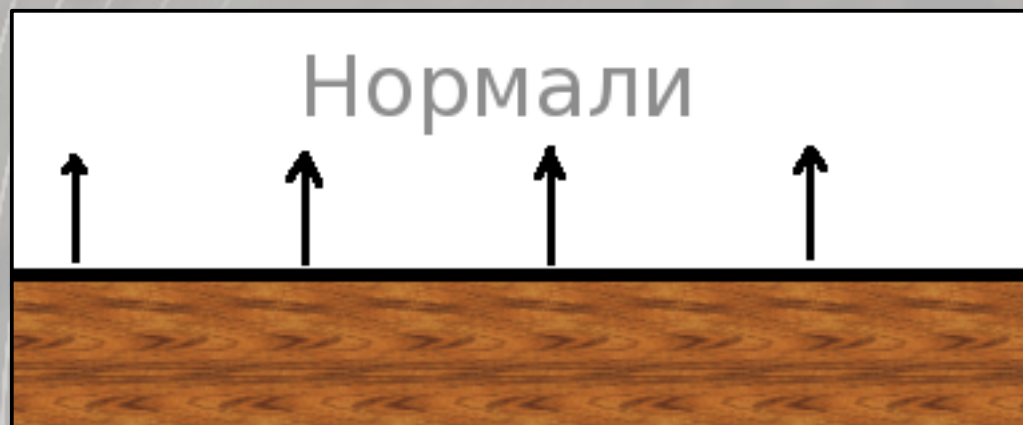
Bump mapping

- 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)



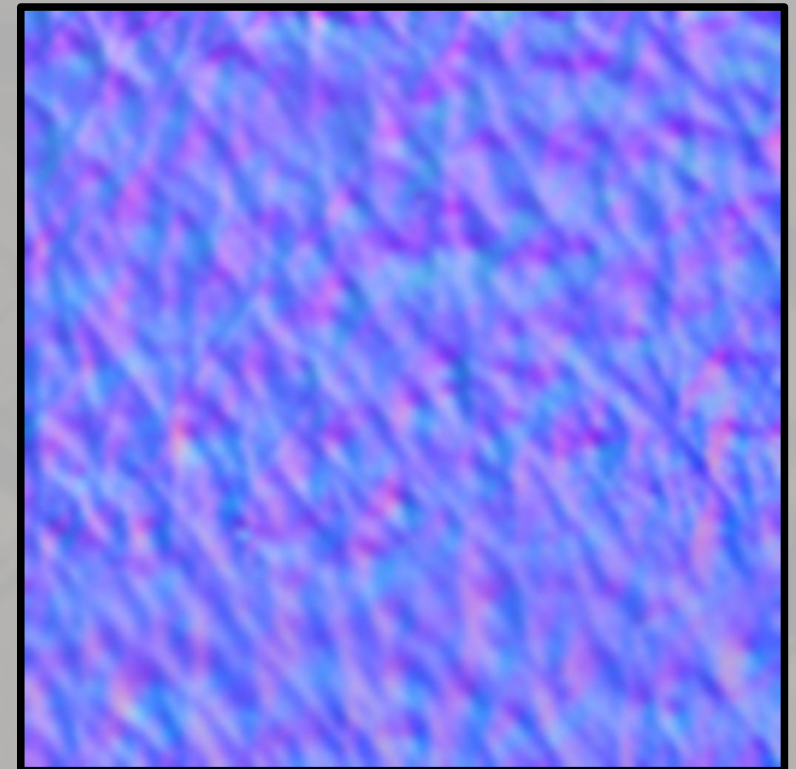
Bump mapping

- 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

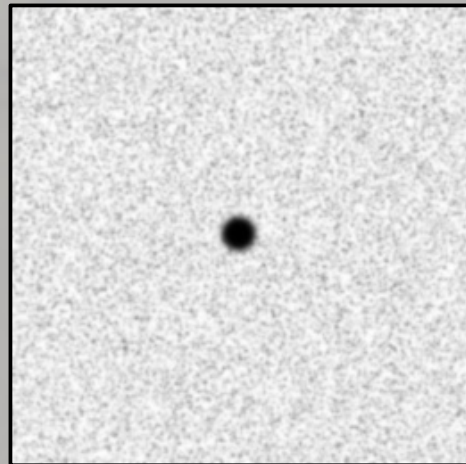
- Вариант 1: текстурата показва с по колко (по x , y , и z направленията) да отклоним нормалния вектор. Т.е., от RGB компонентите на цвета в дадения тексел прехвърляме към XYZ компоненти на отклоняващия вектор
- Пример за такава текстура:



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 :

$$\text{bumpTex}(x, y).dx = \text{origTex}(x, y).intensity() - \text{origTex}(x + 1, y).intensity()$$
$$\text{bumpTex}(x, y).dy = \text{origTex}(x, y).intensity() - \text{origTex}(x, y + 1).intensity()$$

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

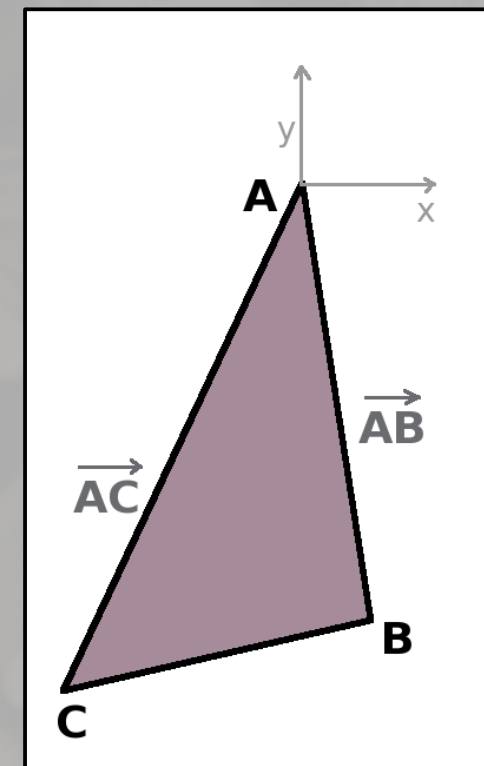
** (изваждаме *ляв-десен*, защото искаме, ако левият пиксел е по-ярък, отклонението да е положително по x (виж схемата 3 слайда назад), т.е. по-ярките пиксели да са по-високи)

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 bwr тар

- 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
 - $x = p_x * AB + q_x * AC$
 - $y = p_y * AB + q_y * AC$



Реализация на bump тар

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

Результати



Витр тар върху други примитиви

- Така разглежданият витр тар 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 витр тар и върху нея
- Вариант 1 на витр тар работи навсякъде, но се изисква специално приготвяне на самия витр тар, 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

