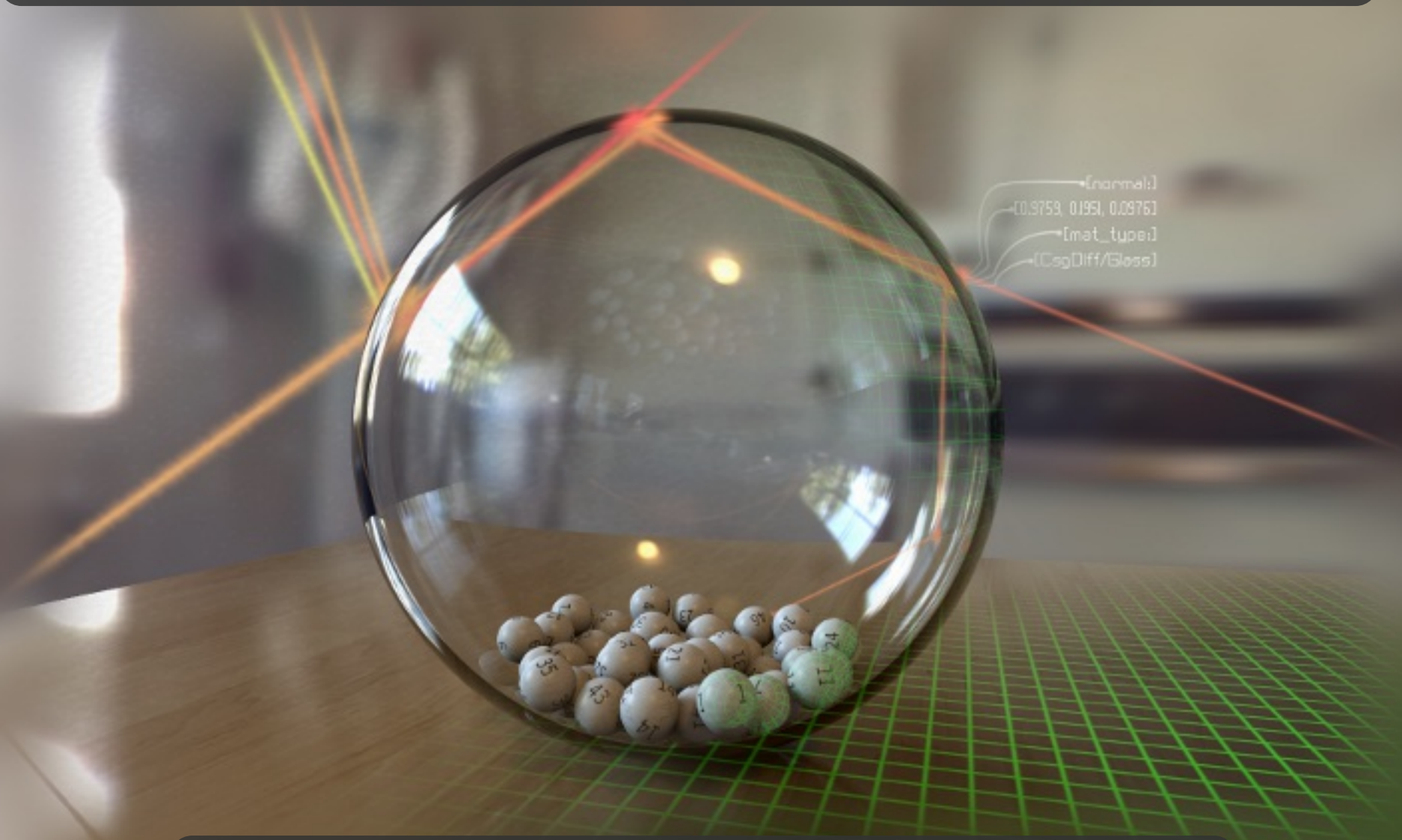


3D графика и трасиране на лъчи v.3.0



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

Тема 7

Монте-Карло методи
Триъгълни мрежи
Нормали

Съдържание

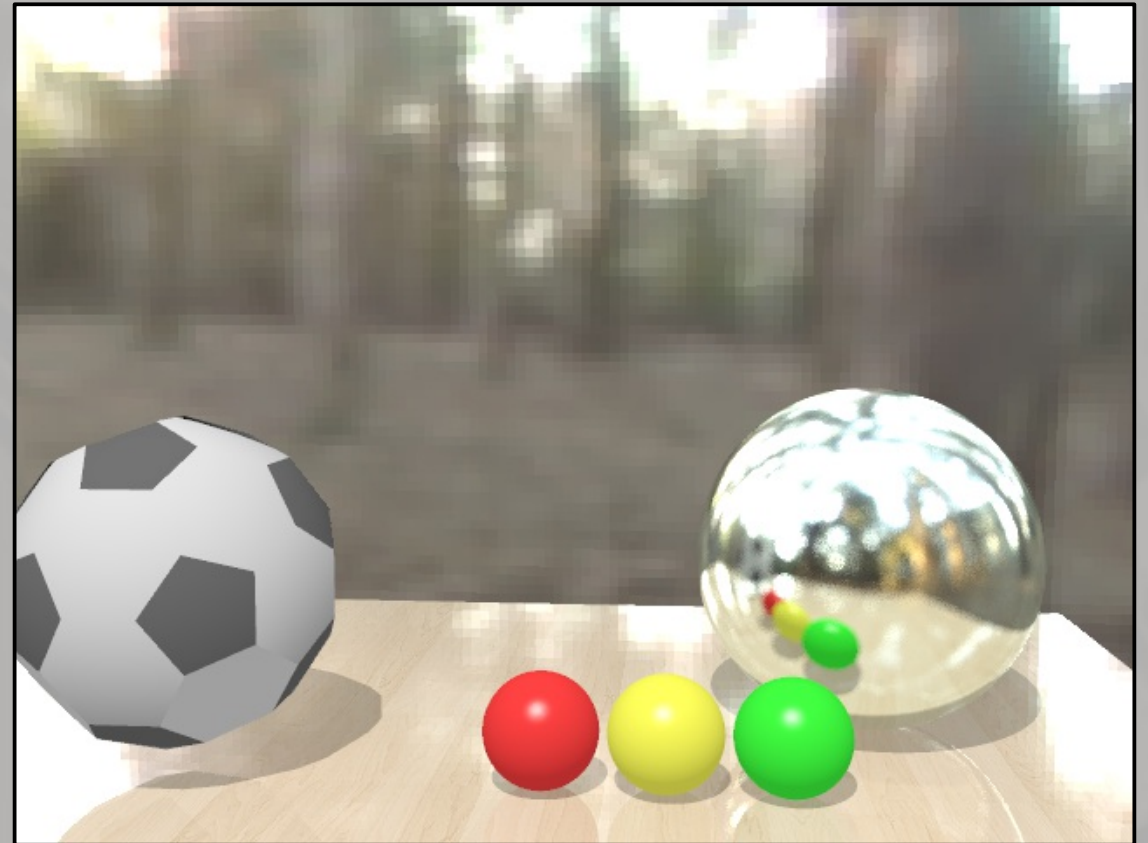
- Анонси
- Нововъведения в учебния рейтрейсър
- Увод в Монте-Карло методите
- Триъгълни мрежи
 - Мотивация
 - Реализация на триъгълните мрежи
 - Пресичане с триъгълник
 - UV координати
 - Нормали в триъгълните мрежи

Анонси

- Тест на 29^{ти} (следващата седмица)
 - По време на първия час, от 17:20ч, продължителност – 20 минути
- Няма да има домашни за тази лекция
- Курсови проекти
 - Условията ще публикуваме във форума от следващата седмица нататък
 - Можете да правите и курсов проект по ваше желание (стига да го одобря :))

Нововъведения в учебния рейтрейсър

- Прогресивно рендериране (може да бъде прекъснато по всяко време)

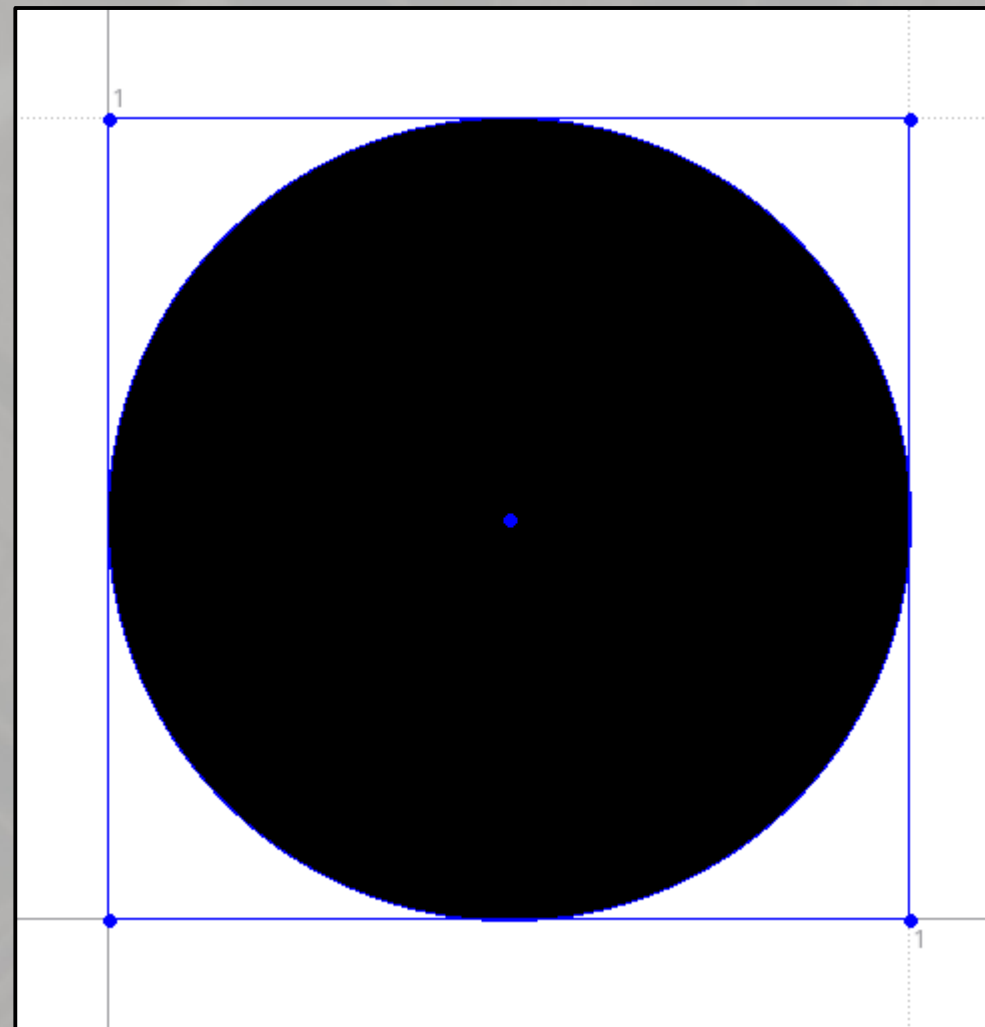


Увод в Монте-Карло методите

- Вече ползвахме Монте-Карло методи за Anti-aliasing и за грапавите отражения
- Най-общо: ако искаме да намерим средната стойност на някаква функция, можем да вземем N случайни параметъра, да сметнем функцията в тях, и да усредним
 - Anti-aliasing: $f(\Delta x, \Delta y) := \text{raytrace}(\text{getScreenRay}(x + \Delta x, y + \Delta y))$
 - Деф. област: $0 \leq \Delta x, \Delta y \leq 1$
 - Glossy: $f(dx, dy) := \text{raytrace}(\text{reflect}(\text{ray}, \text{mangle_normal}(dx, dy)))$
 - Деф. област: $dx^2 + dy^2 \leq \tan((1 - \text{glossiness}) * \pi/2)^2$

Пример за прост Монте-Карло метод

- Ще пресметнем π чрез отношението между площите на кръг и описания около него квадрат
- $S_{\text{disc}} = \pi r^2 = \pi * 0.5^2 = \pi/4$
- $S_{\text{box}} = a * a = 1$
- $f(x, y) := ((x - 1/2)^2 + (y - 1/2)^2 \leq 1/4) ? 1 : 0.$



По-„умна“ задача

- Три точки са избрани случайно и равномерно в единичния квадрат. Каква е вероятността, триъгълникът, образуван от тези точки, да е остър?
- Имаме осветление от правоъгълна лампа. Каква част от лампата се „вижда“ в дадена точка от сцената?

Монте-Карло интегриране

- Монте-Карло интегриране:

$$\int_{\Omega} f(x) dx \approx \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{\rho(x_i)}$$

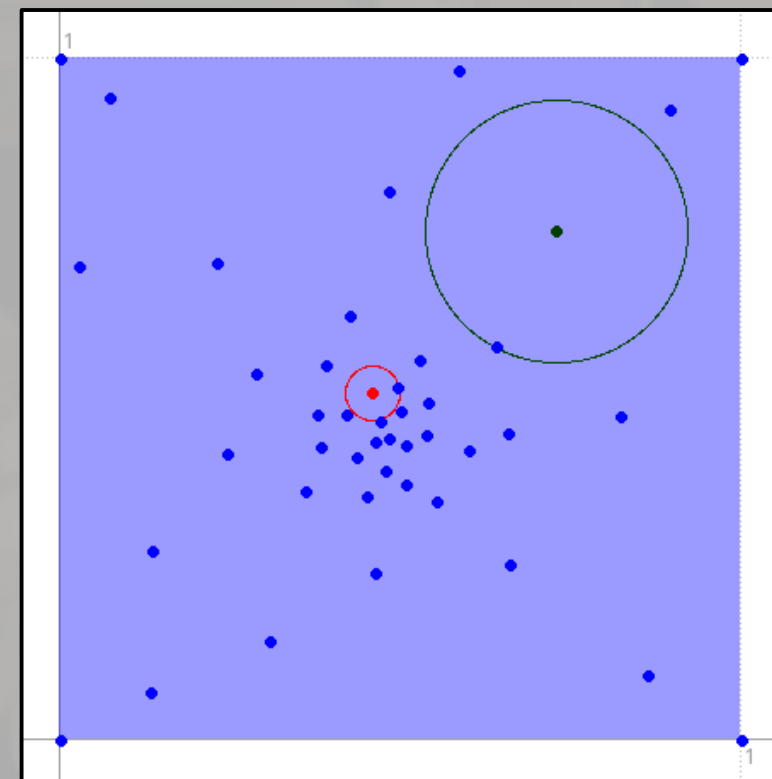
- $f(x)$ е функцията, чиито интеграл искаме да пресметнем
- Ω е областта, в която пресмятаме определения интеграл
- x_i са случайните точки, които random генератора избира
- $\rho(x_i)$ е вероятността точката x_i да бъде изтеглена
- Грешката е пропорционална на $1/\sqrt{N}$

Монте-Карло интегриране

- Функцията $\rho(x)$ нормира резултата, като взема под внимание формата на Ω , както и неравномерността на генерираните случайни точки
 - $\rho(x)$ е в смисъла на вероятностна плътност на генератора:
 - Когато генерираме равномерно-разпределени точки в единчината отсечка/квадрат, то $\rho(x) = 1$ навсякъде
 - Ако генерирахме случайни точки в $[0..5]$, $\rho(x)$ щеше да е 0.2.
 - $x \text{ in } [0..2] \times [0..2] \rightarrow \rho(x) = 0.25$
 - Можем да ползваме неравномерен генератор за техниката `importance sampling`

Importance sampling

- Можем да контролираме разпределението на случайния генератор и да искаме точките да са по-“нагъсто” в областите, в които функцията е „интересна“
 - Например: по-гъсто семплиране в средата на квадрата
 - За сивата точка, $\rho(x)$ ще е по-ниско, и точката ще участва с по-голяма тежест в сумата; това отразява факта, че точката е представител на по-голямо „кръгче“ около нея, в което няма други точки

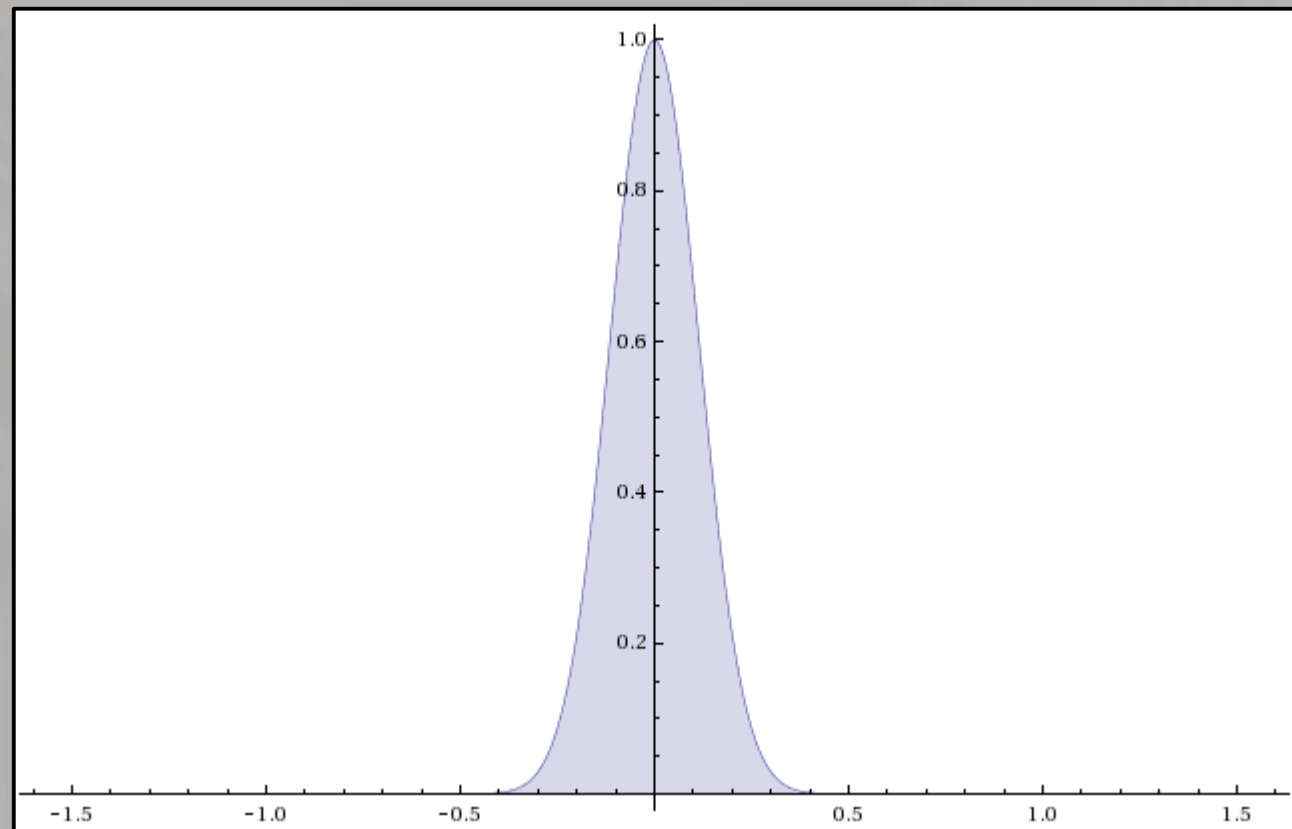


Importance sampling - пример

- Например, искаме да сметнем средната мощност на Phong отражение при конкретен експонент (77)

т.е.,
$$\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos(x)^{77} dx$$

- $\Omega = [-\pi/2 .. \pi/2]$
- $f(x) := \cos(x)^{77}$
- $\rho(x) = 1/\pi$
(без importance sampling)



Importance sampling - пример

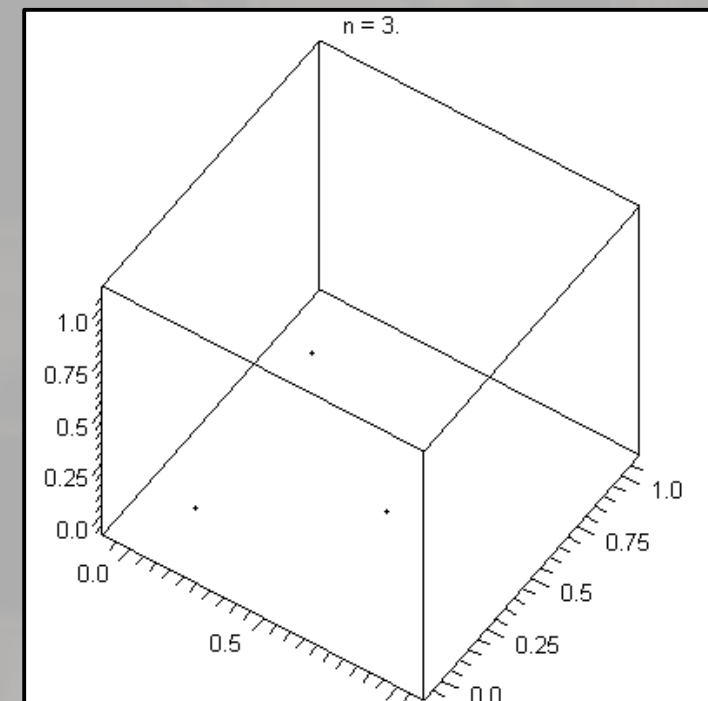
- Вариант 2: ползваме генератор на числа с нормално разпределение ($\mu = 0, \sigma = 0.15$)
 - σ е избрано така, че да се припокрива с „активната“ зона на $\cos(x)^{77}$

$$\rho(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2}{2\sigma^2}}$$

- Новото семплиране дава същия резултат като равномерното, но е по-бързо сходящо

Свойства на Монте-Карло методите

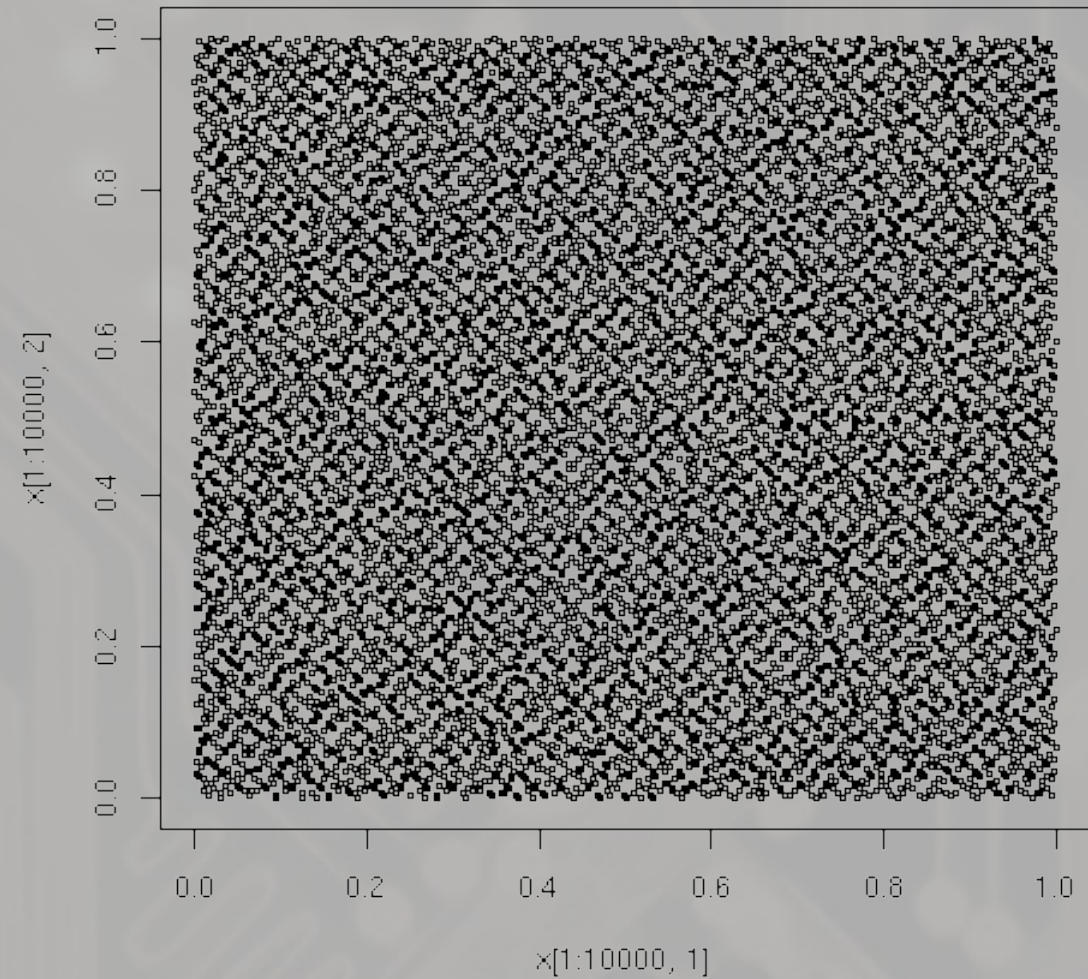
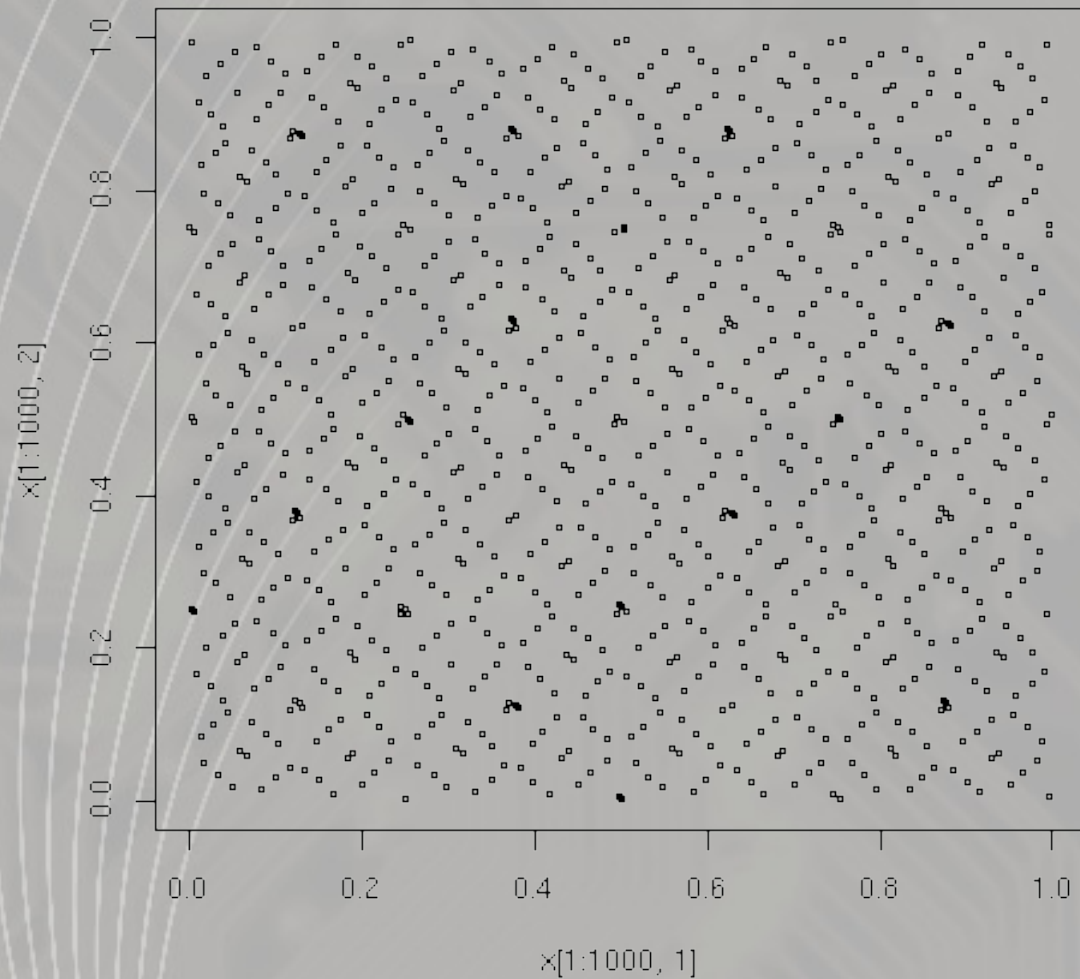
- Монте-Карло симулациите са силно чувствителни към качествата на случайния генератор
 - LSPRNG (rand() и колеги) се представят плачевно при по-високи размерности
 - Ще ползваме Mersenne Twister по-нататък
- Грешката на метода зависи основно от броя семпли, N
 - Може да се абстрахираме от „the curse of dimensionality“



Оптимизации на Монте-Карло методите

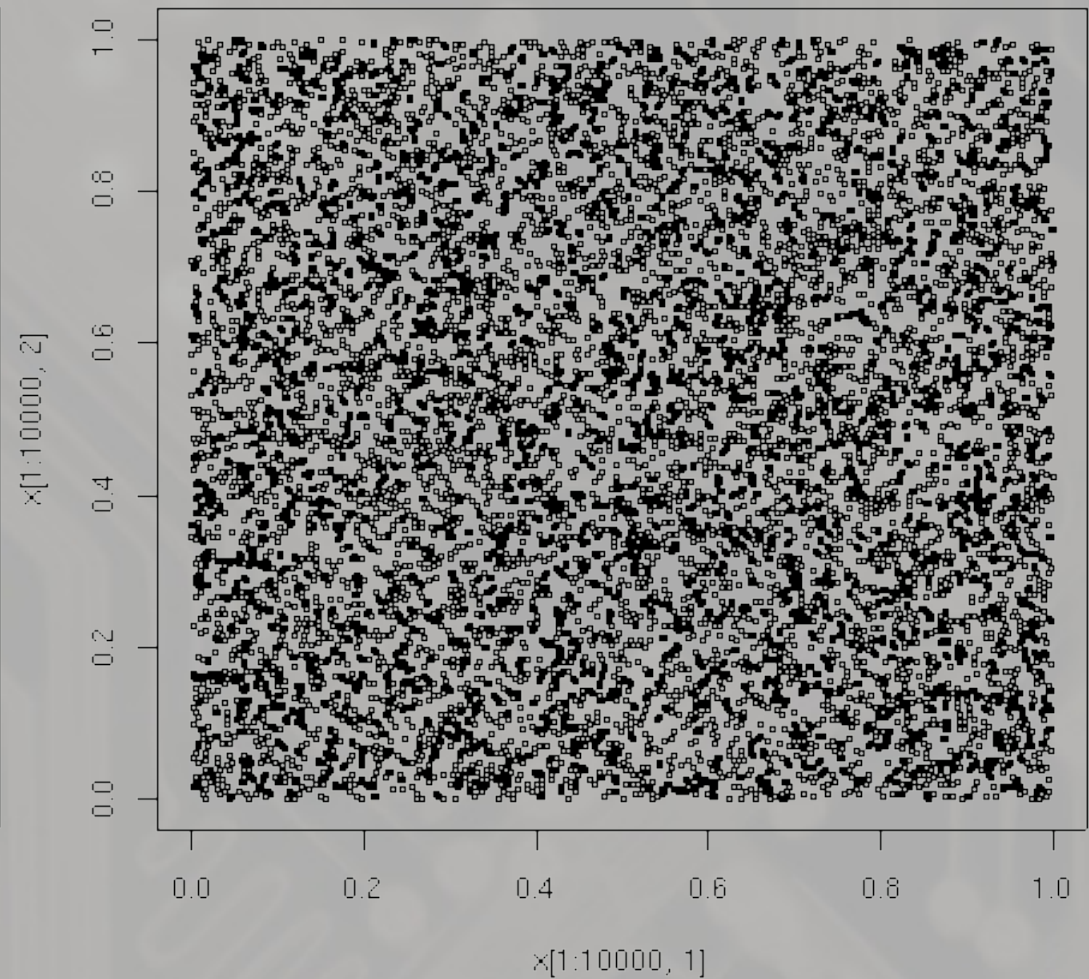
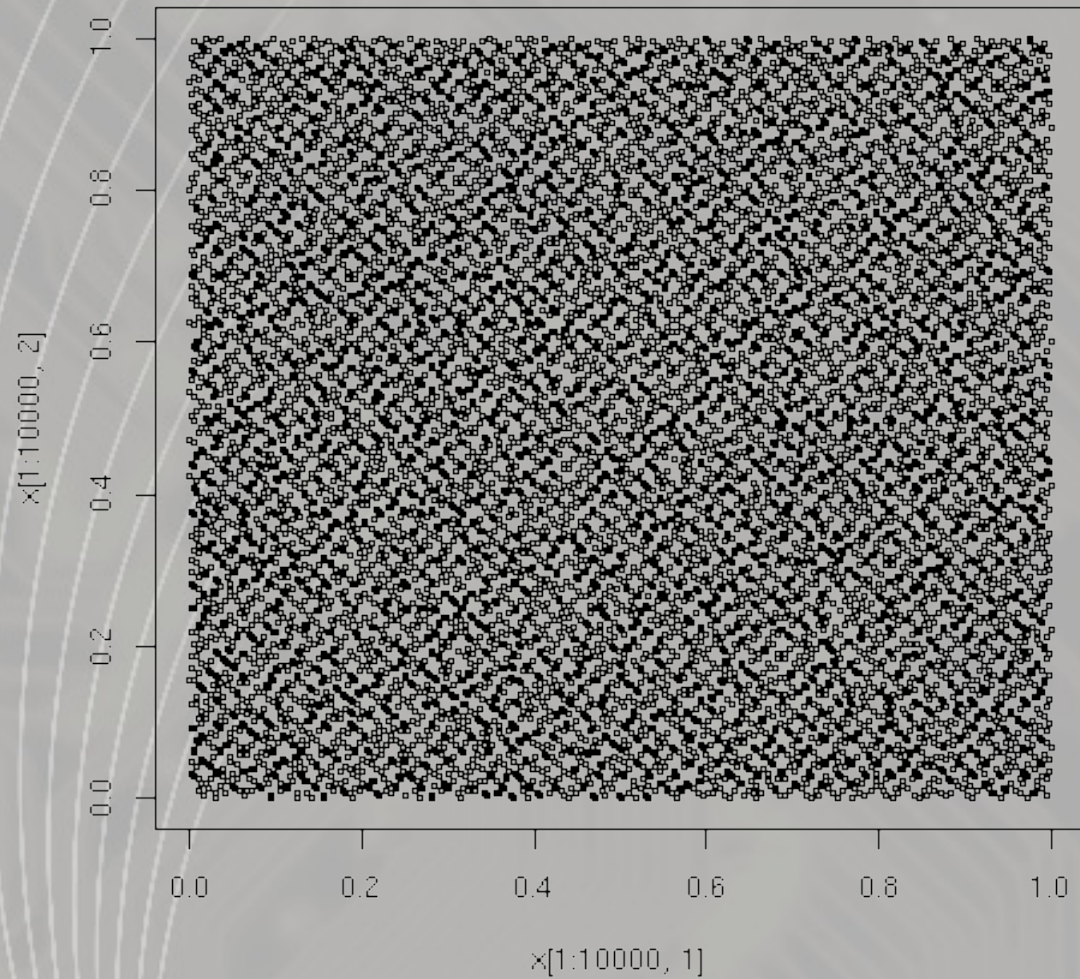
- Stratified sampling
 - Ще дадем пример в лекция 10
- Low-discrepancy sequences
 - Чисто случайните точки често не запълват пространството „добре“ (струпвания, големи „дупки“, ...)
 - LDS приличат на случайни редици, но са напълно детерминистични, и гарантират „плътно“ / „равномерно“ запълване на пространството (= по-малко шум при равен брой семпли)
 - Монте-Карло метод, ползващ LDS, се нарича QMC (квази-Монте-Карло)

Low-discrepancy sequences - пример



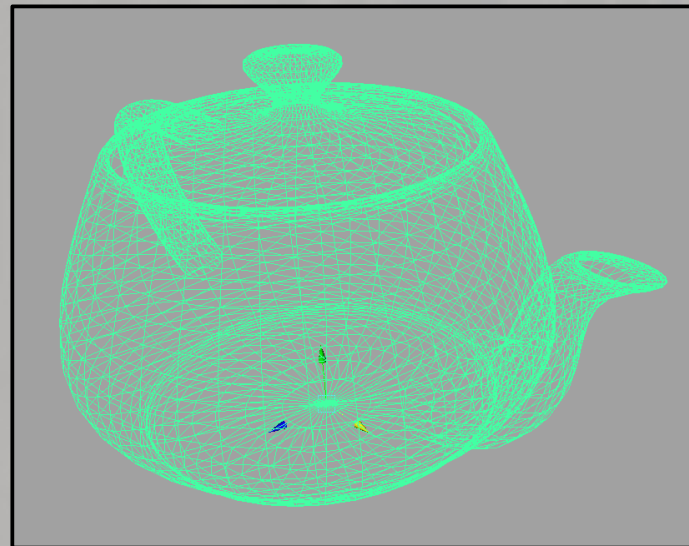
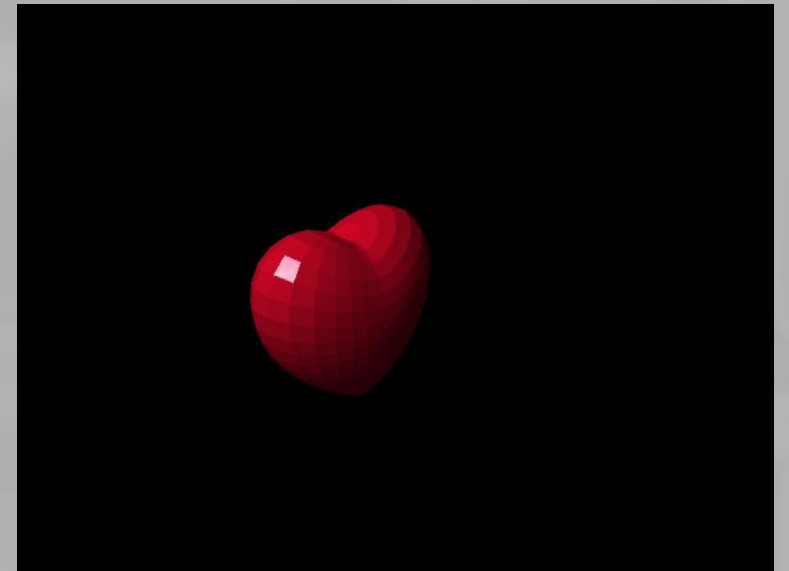
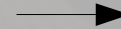
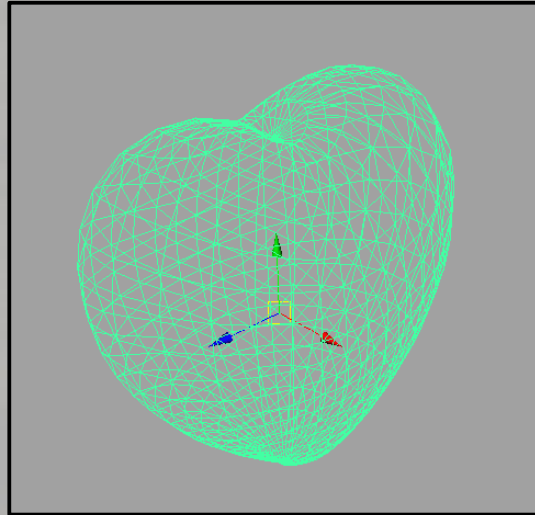
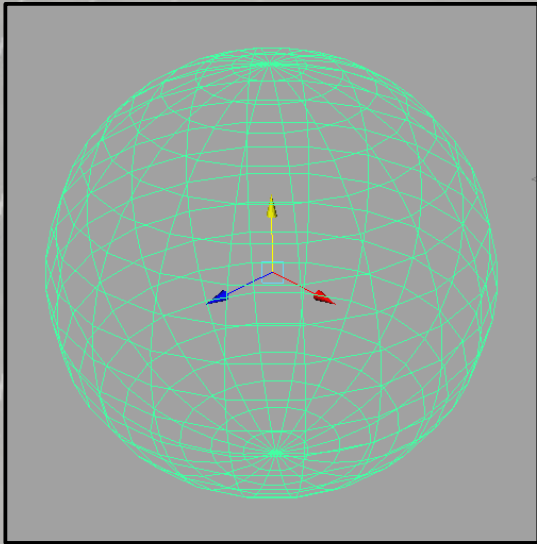
- Двумерно LDS с 1000 (ляво) и 10000 (дясно) точки

Low-discrepancy sequences - пример



- LSD срещу чисто случайна редица

Триъгълни мрежи

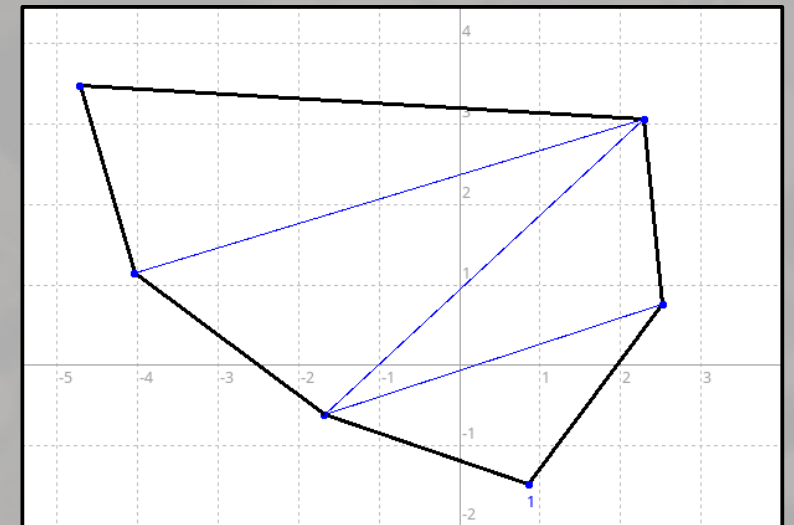
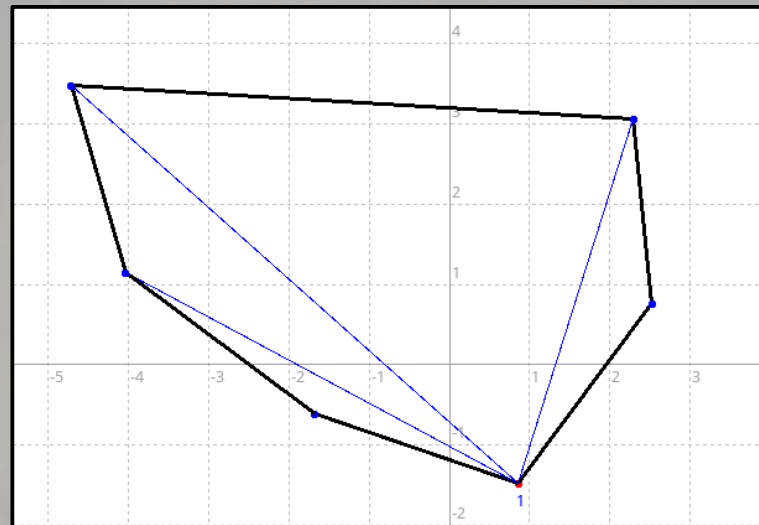
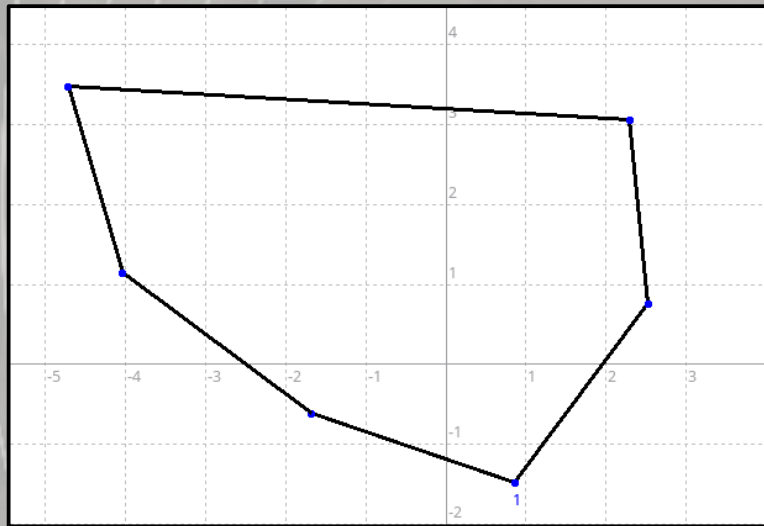


Триъгълни мрежи

- Триъгълните мрежи са нещо универсално в съвременната компютърна графика
 - Z-buffer разчита изцяло на тях
- Основната идея: всички повърхности на 3D обектите да се нацепят на малки многоъгълници (с пасващи си страни и върхове между съседните многоъгълници)
- Често имаме контрол върху броя многоъгълници
 - Може да изберем колко фина да е мрежата; компромис между време за рендериране и качество

Терминология

- Mesh = триъгълна мрежа
- Триъгълник \approx многоъгълник (полигон)
 - Изпъкналите многоъгълници се триангулират лесно



Мотивация за триъгълните мрежи

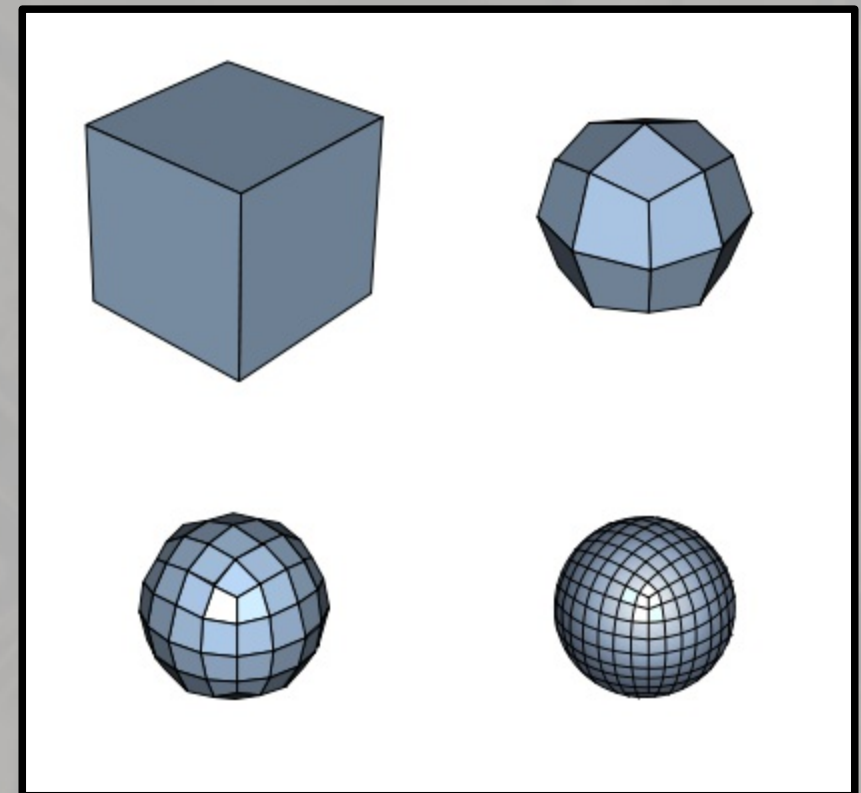
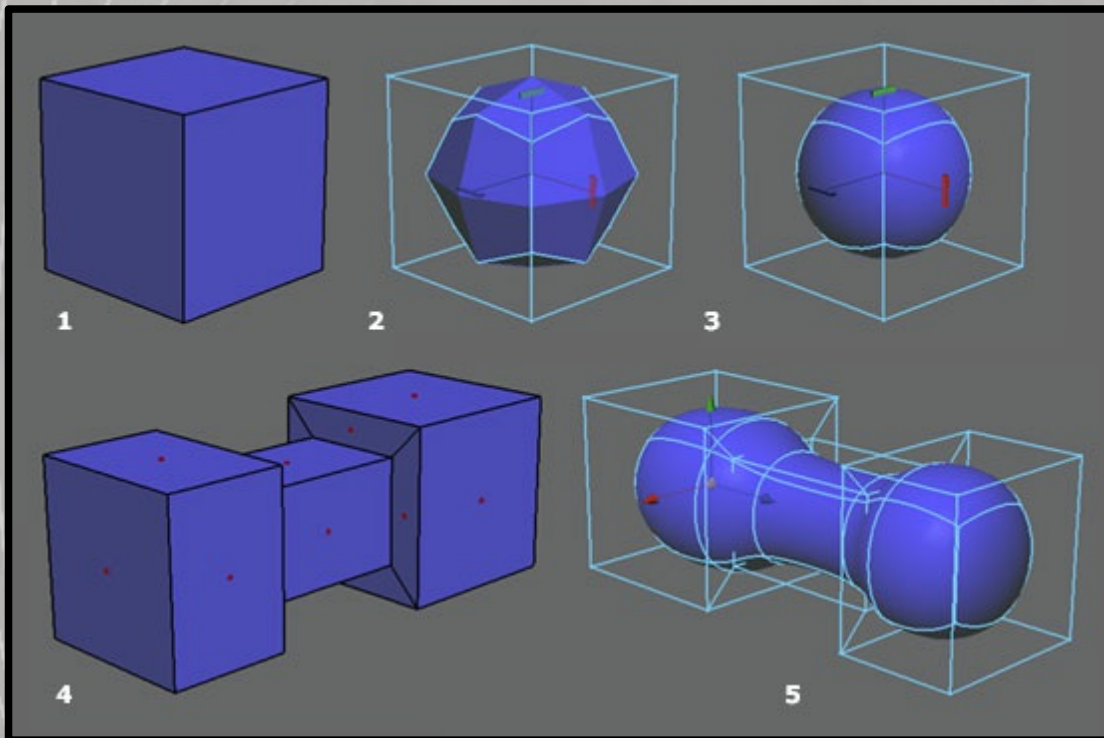
- Прости са (триъгълника е една от най-простите 3D фигури)
- Могат да наподобяват произволно добре практически всякаква друга геометрия
- Позволяват preview (по време на редактиране например), чрез Z-buffer или wireframe алгоритми
- Редактирането е лесно и доста интуитивно

Примери за редактиране

- Местене на връх/върхове, ребра, триъгълници...
- Мащабиране и завъртане в произволна посока
- Рязане и цепене на триъгълници
- Операции като extrude (пресоване), огъване, ...
- Subdivision surfaces
 - Това е метод, при който от груба начална мрежа, чрез някакъв алгоритъм, се генерира нова триъгълна мрежа, която е по-фина от старата и представлява „загладена“ версия на старата мрежа

Subdivision surfaces

- Алгоритмите за subdivision са относително прости, но позволяват артистите да постигнат доста детайлен и прецизен обект без да се трудят много (започват от много груб модел)



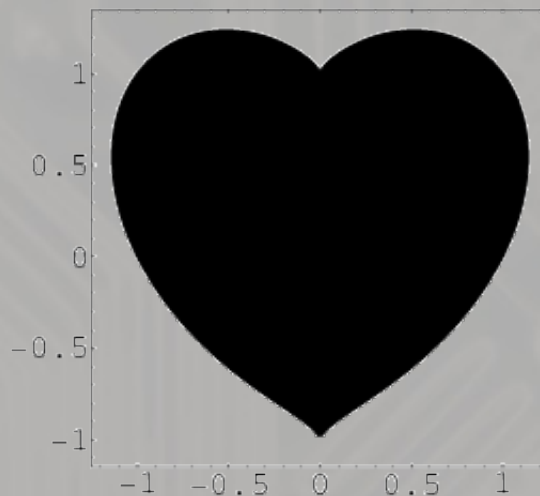
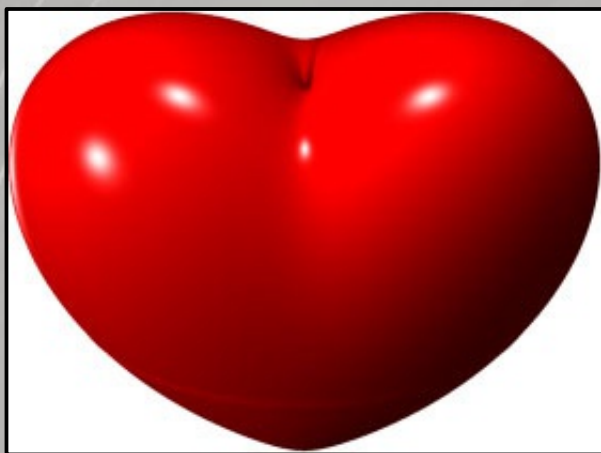
Как се създават триъгълните мрежи?

- Явни формули – за геометричните обекти като сфера, цилиндър, тор, куб, пирамида, призма...
 - Някои от тях са гладки, други не – триъгълните мрежи поддържат и двата вида повърхности – ще дадем пример по-нататък
- Неявни формули
 - Наричани още Isosurfaces, implicit surfaces и прочее
 - Повърхността се задават с формула от вида $f(x, y, z) = 0$, т.е. при дадена 3D функция, повърхността се състои от точките в пространството, за които функцията е 0

Как се създават триъгълните мрежи?

- Например, $f(x, y, z) = x^2 + y^2 + z^2 - 1$ задава сфера с радиус 1
- По-сложните формули позволяват да се създават интересни форми: ето пример за повърхност с формата на сърце (изпъкнал кардиоид):

$$f(x, y, z) = (x^2 + \frac{9}{4}y^2 + z^2 - 1)^3 - x^2z^3 - \frac{9}{80}y^2z^3 = 0$$



Как се създават триъгълните мрежи?

- Isosurface по принцип могат да се пресичат с лъчи, но обикновено е доста по-просто да се превърнат в триъгълни мрежи
 - Чрез алгоритъма „Marching Cubes“
- Сплайни
 - Има програми, позволяващи създаването на 3D обекти, дефинирани като сплайни
 - Например, отделните детайли на една кола се моделират лесно като сплайн кръпки
 - Всъщност сплайните са измислени именно с тази цел!

Как се създават триъгълните мрежи?

- Трасирането на сплайни в 3D обаче е неудобно, затова често при самото рендериране, повърхностите се превръщат до триъгълни мрежи
- Може обектите да се моделират като мрежи още от самото начало
 - Чрез 3ds Max, Maya, Blender...
 - Един пример [[youtube](#)]

Как се създават триъгълните мрежи?

- Триъгълни мрежи могат да се създават и от релефни карти
 - Релефните карти представляват двумерна текстура, показваща „височината“ на някаква повърхност
 - Може да отговарят на истински релеф някъде из Земята (топографска карта), или изкуствено моделирани
 - Релефните карти също може да се трасират, но могат да се превърнат и в триъгълни мрежи (всеки пиксел от картата става връх в мрежата)

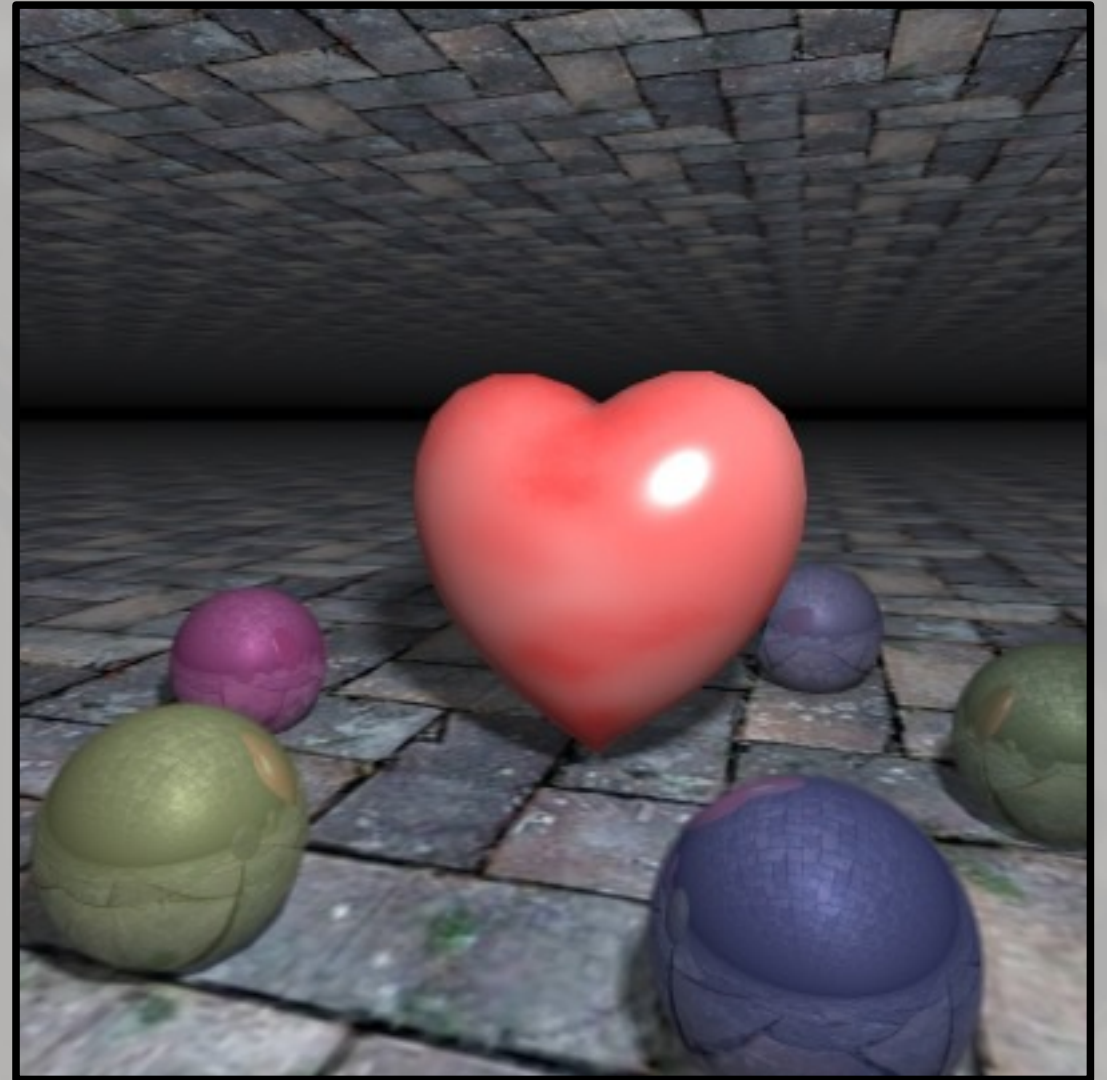
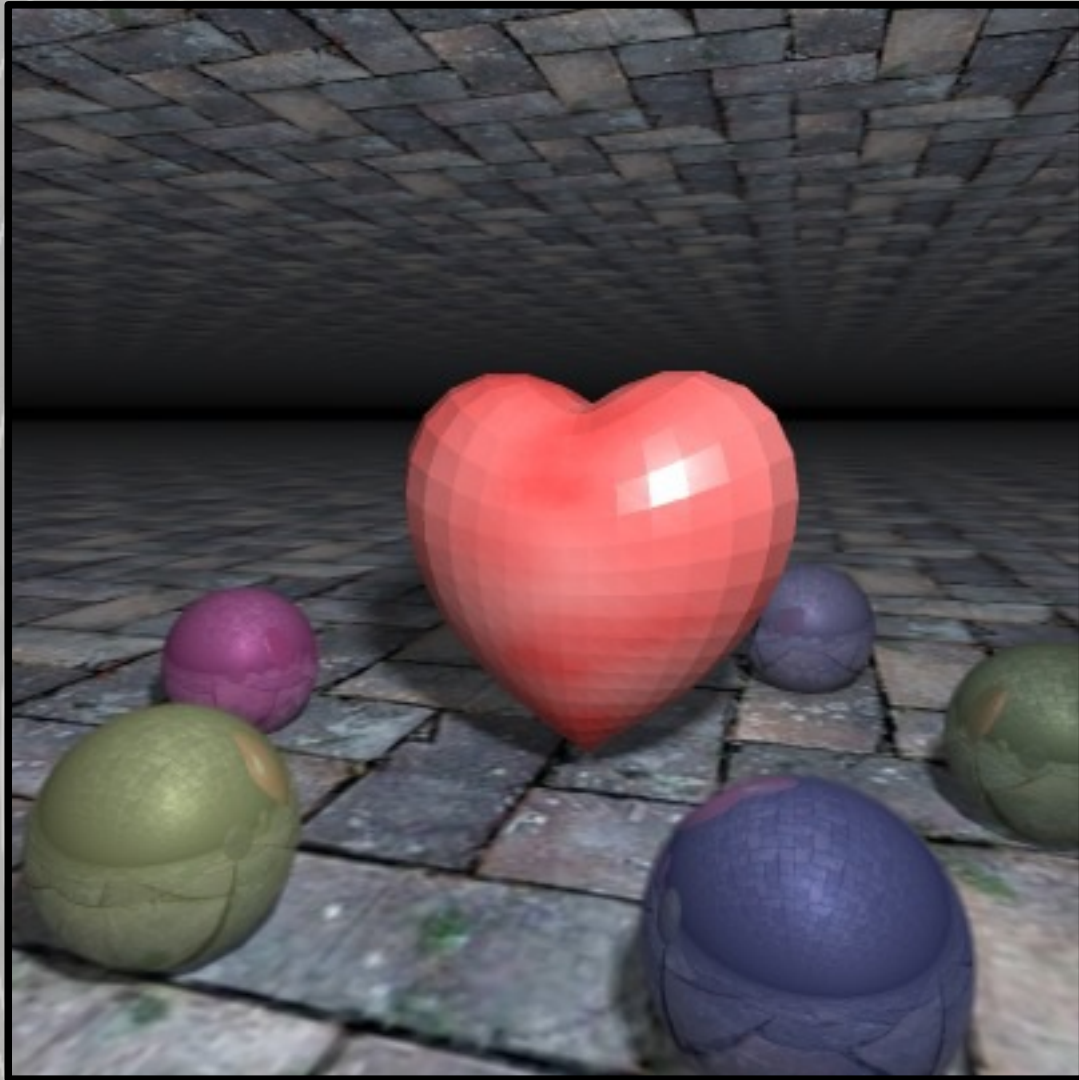
Как се създават триъгълните мрежи?

- Съществуват още много начини за създаване на триъгълни мрежи
- Не е нужно човек да създава всичко сам, има сайтове с безплатни 3D модели

Свойства на триъгълната мрежа

- Само повърхности vs обекти с обем
- Херметически затворени vs отворени обекти
 - Например, цилиндър, сфера, куб са затворени, докато чайникът не е
- Гладки vs твърди стени
 - Мрежите са фундаментално твърди (фасетни), но могат да се направят да изглеждат гладки чрез подходяща интерполация на нормалите вътре в триъгълниците
 - Това изисква да пазим по един нормален вектор във всеки връх (и те да са различни)

Свойства на триъгълната мрежа



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

Представяне на триъгълната мрежа

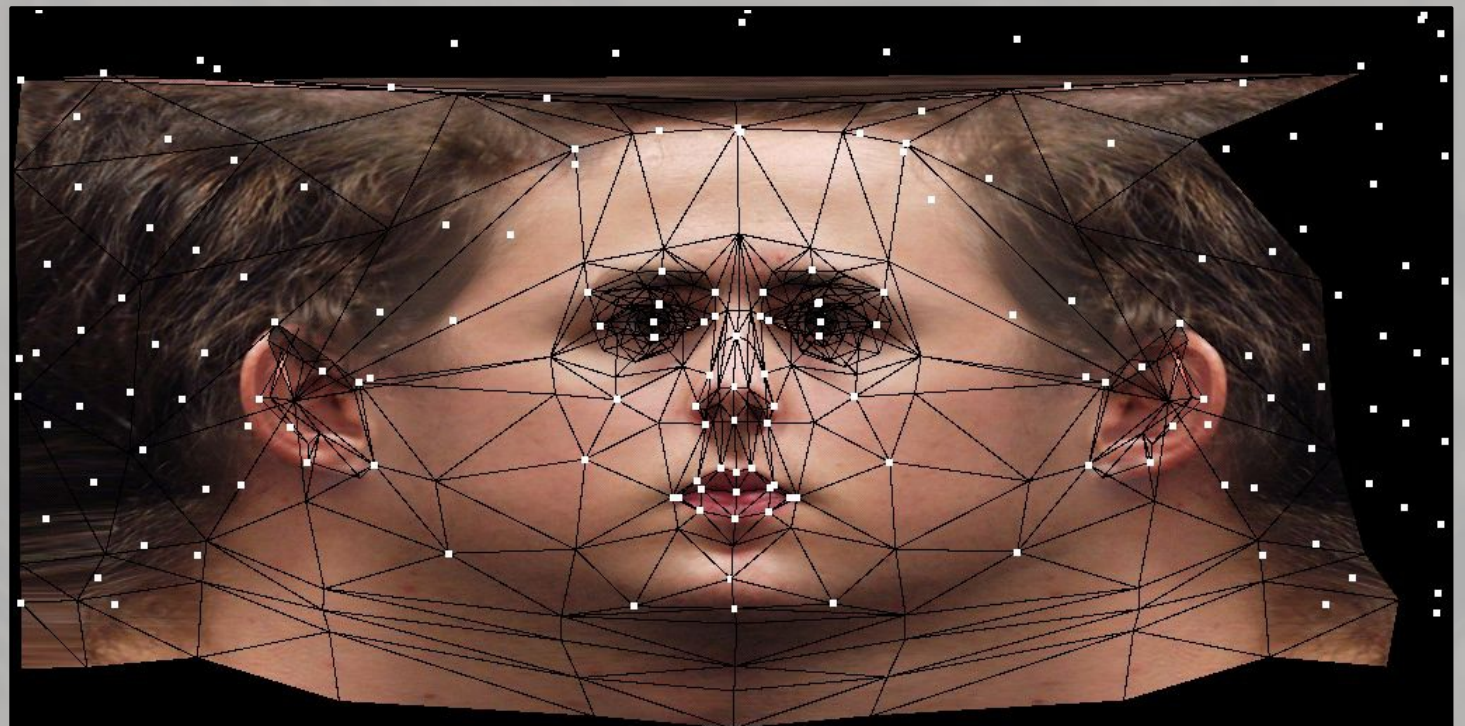
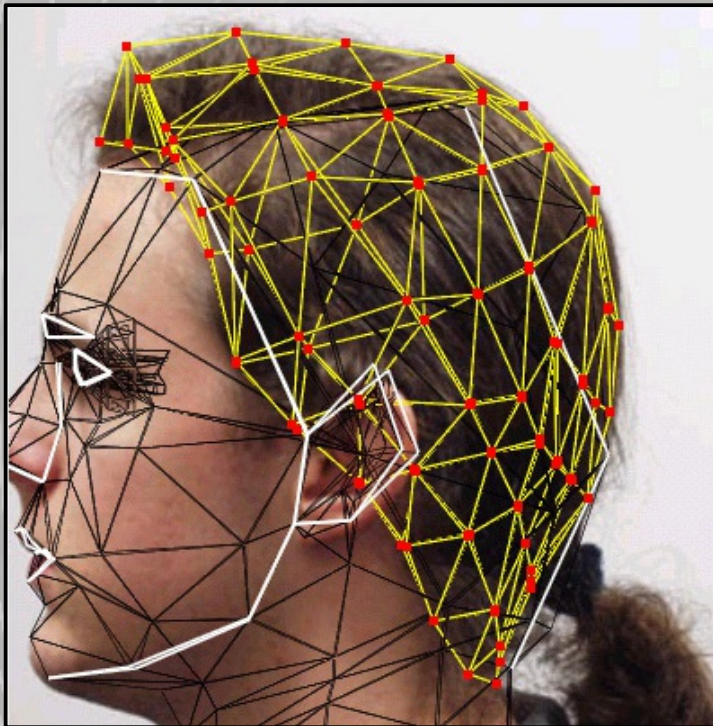
```
struct Triangle {  
    int v[3], n[3], t[3];  
    Vector gnormal; // geometric normal of the triangle itself  
};  
  
class Mesh: public Geometry {  
    vector<Vector> vertices;  
    vector<Vector> normals;  
    vector<Point> uvs;  
    vector<Triangle> triangles;  
    ...  
};
```

Представяне на триъгълната мрежа

- Т.е. за всеки триъгълник имаме:
 - Три върха, които дефинират положението му в пространството
 - Един „истински“ нормален вектор (пресметнат от 3-те върха)
 - Три нормални вектора, за всеки от върховете
 - Потенциално различни, особено за гладки повърхности
 - Показват нормалата на повърхността в тази точка
 - Интерполират се между трите върха; ще демонстрираме по-нататък
 - Три UV координати (за текстуриране)
 - Също се интерполират между трите върха

Текстурни (UV) координати

- Идеята е, че всеки 3D триъгълник съответства на 2D триъгълник в текстурното пространство
- Т.е. всеки 3D триъгълник може да бъде облепен с произволен триъгълен отрязък от текстурата.



Интерполация в триъгълник

- Споменахме, че ще ни трябва интерполация в триъгълник; само че как точно ще я реализираме?
- Барицентрични координати: нека имаме триъгълник ABC и произволна точка P в него; барицентричните координати на точката P са числата $\lambda_1, \lambda_2, \lambda_3$, за които:

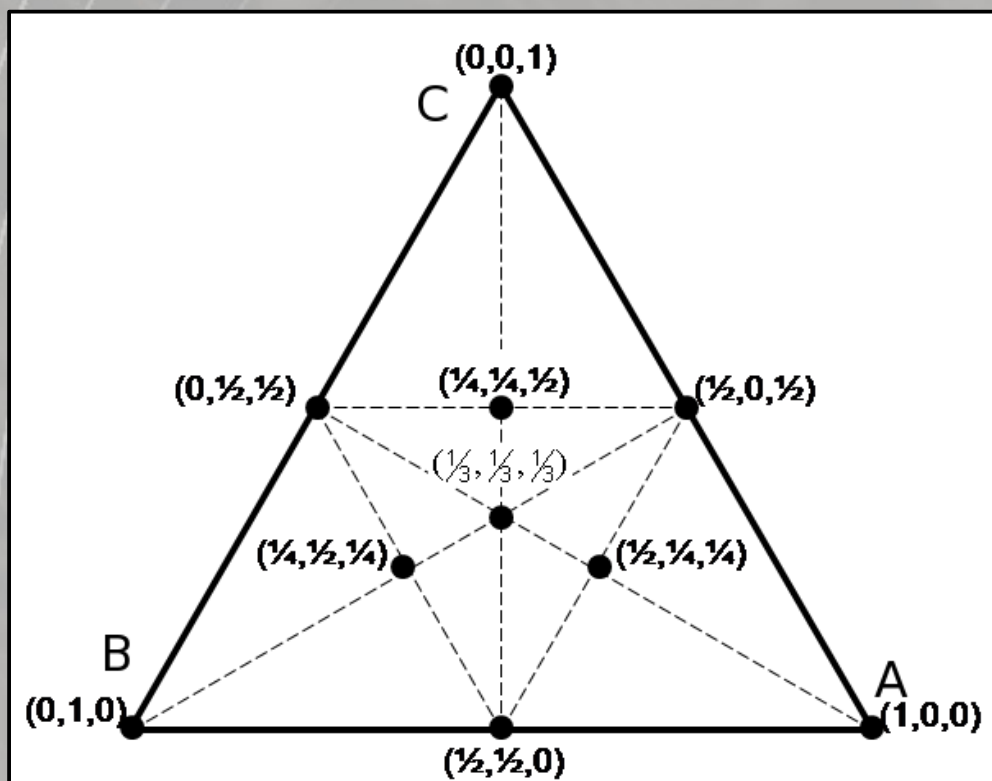
$$P = \lambda_1 A + \lambda_2 B + \lambda_3 C,$$

и е изпълнено че:

$$0 \leq \lambda_1, \lambda_2, \lambda_3 \leq 1$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

Барицентрични координати



- На практика може да работим само с λ_2 и λ_3 , тъй като $\lambda_1 = 1 - \lambda_2 - \lambda_3$.
- Нова дефиниция:
$$P = A + (B-A)\lambda_2 + (C-A)\lambda_3 \quad (*)$$
$$0 \leq \lambda_2, \lambda_3 \leq 1$$
$$\lambda_2 + \lambda_3 \leq 1$$
- Т.е. дефинирахме нещо като координатна система

Интерполация в триъгълник

- Ако във формулата (*) вместо точките A, B, C заместим със съответните нормали в трите върха, и имаме конкретни λ_2, λ_3 , то интерполираме между нормалите в трите върха
- Аналогично, ако вместо A, B, C заместим UV координатите в трите върха, то интерполираме между UV координатите

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

- Три точки в пространството лежат в една равнина
- Следователно може да сведем задачата за пресичане на лъч с триъгълник до такава за пресичане на лъч с равнина
 - Само че ни интересува пресечната точка да е точно в отрязъка на равнината, в който се намира триъгълника
 - За целта, ще намерим барицентричните координати на пресечната точка на лъча с равнината, и ще проверим дали изпълняват условията за барицентрични координати

Пресичане с триъгълник

- Дадено:
 - А, В, С – върхове на триъгълника
 - О – 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:

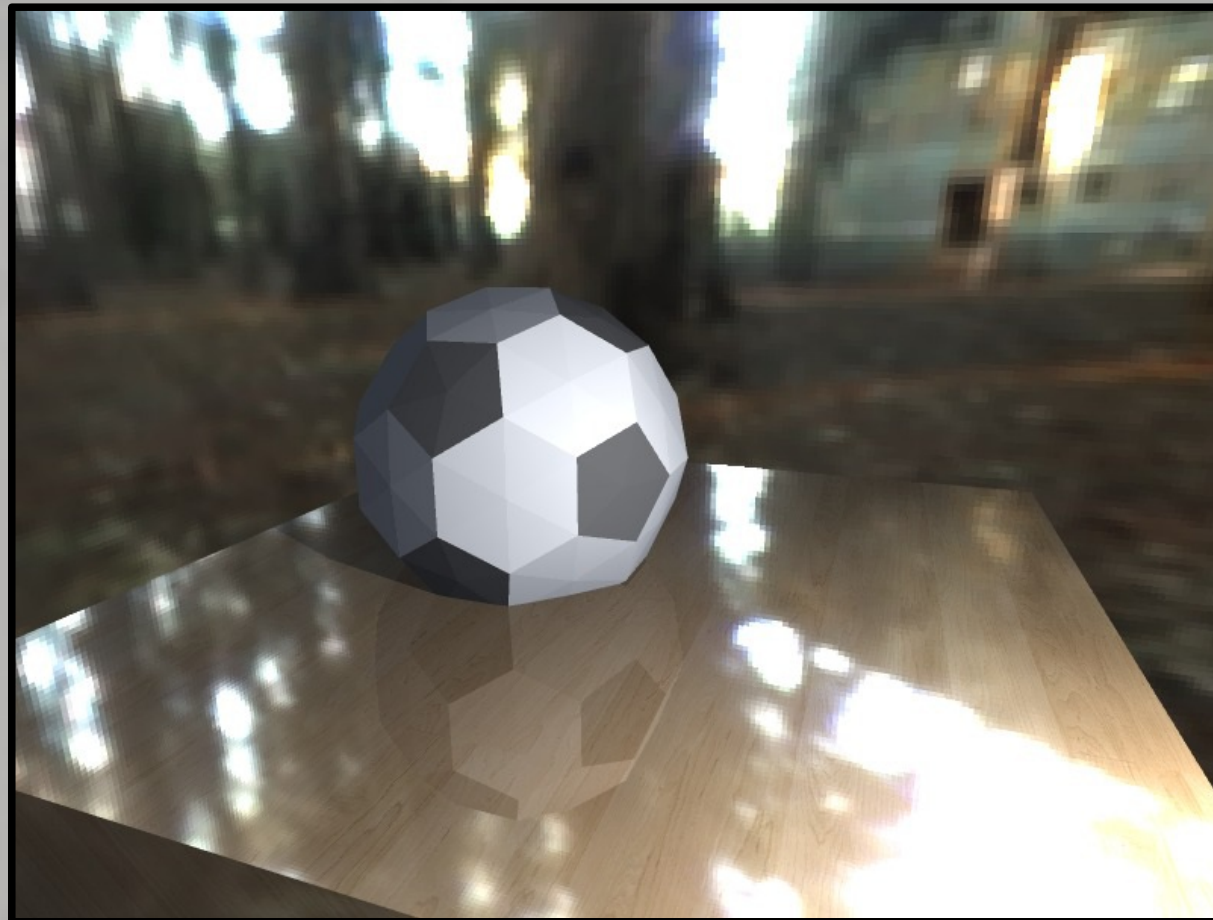
$$\text{Det} = (a \wedge b) * c$$

Решение на системата

- $\text{Det}_{\text{cr}} = ((B-A) \wedge (C-A)) * -D$
- Нека $H = (O-A)$
 $\lambda_2 = ((H \wedge (C-A)) * -D) / \text{Det}_{\text{cr}}$
 $\lambda_3 = (((B-A) \wedge H) * -D) / \text{Det}_{\text{cr}}$
 $\gamma = (((B-A) \wedge (C-A)) * H) / \text{Det}_{\text{cr}}$

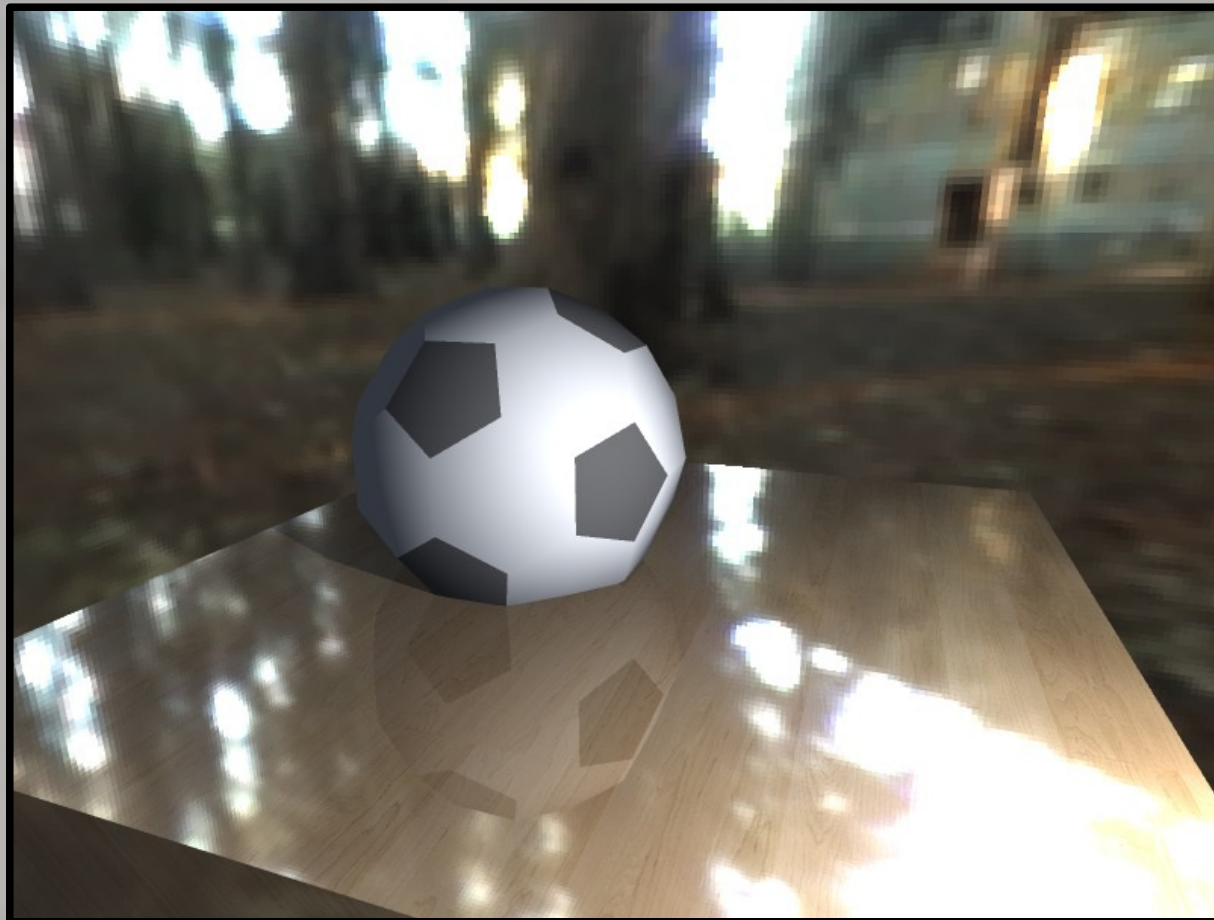
Резултат

- Ако в `info->norm` слагаме геометричния нормал на триъгълника, т.е. не интерполираме нормалите:



Резултат

- Интерполиране на нормалите:
$$n = \text{normalize}(nA + \lambda_2 * (nB - nA) + \lambda_3 * (nC - nA))$$



Оптимизации

- Ако повърхнината е затворена херметически, може да пропускаме триъгълниците, за които $(\mathbf{gnormal} * \mathbf{ray.dir} > 0)$. Те са от „задната“ страна на триъгълната мрежа и няма шанс да ги пресечем, освен ако не тръгваме отвътре
- Може да опишем сфера или куб около триъгълната мрежа и да проверим в самото начало дали лъча пресича тази описваща геометрия. Ако не пресича нея, значи не пресича и мрежата.