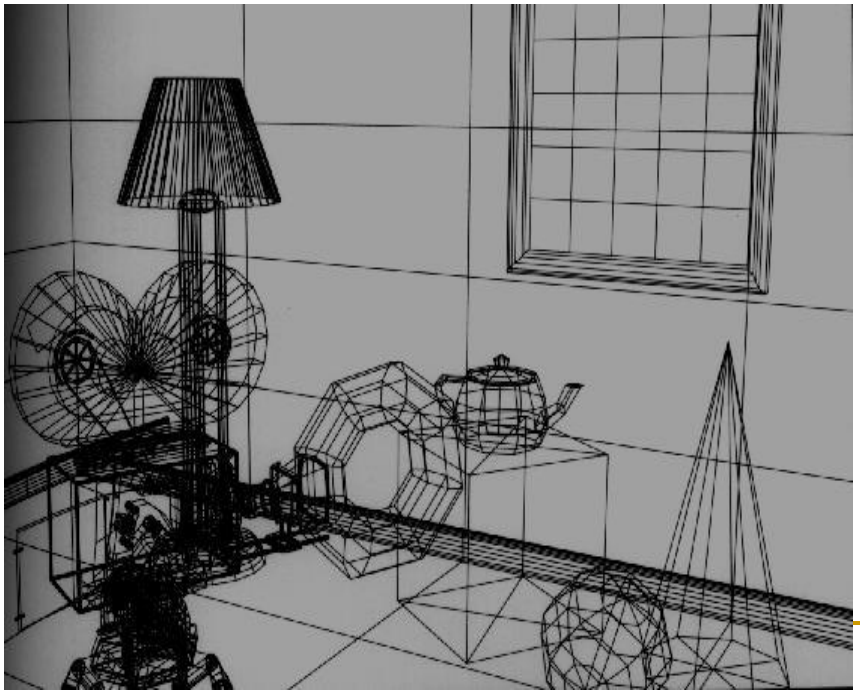
Компютърна графика

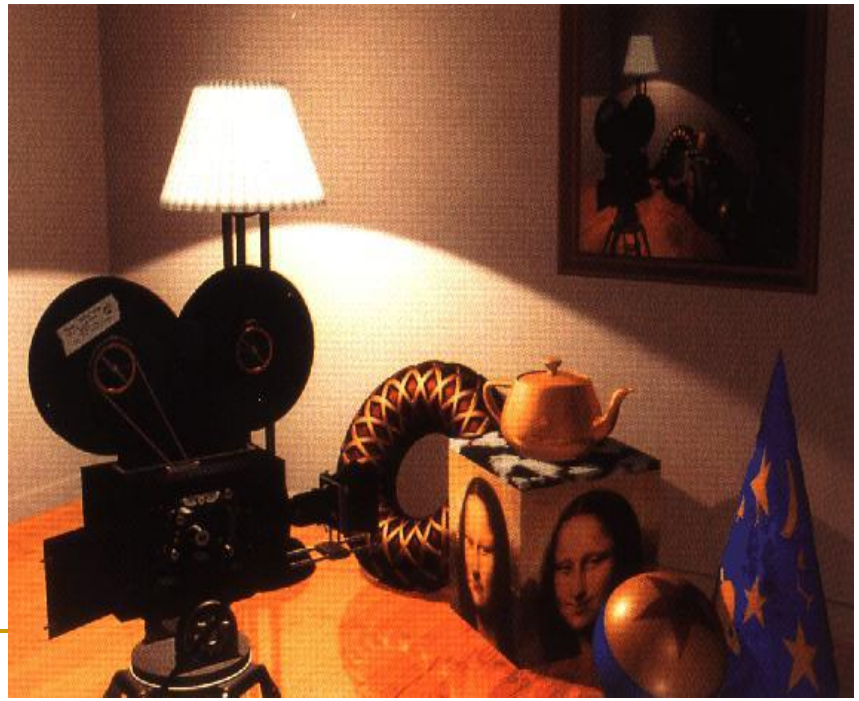
Геометрично моделиране на 2D и 3D обекти

доц. Милена Лазарова, кат. КС, ФКСУ

Геометрично моделиране

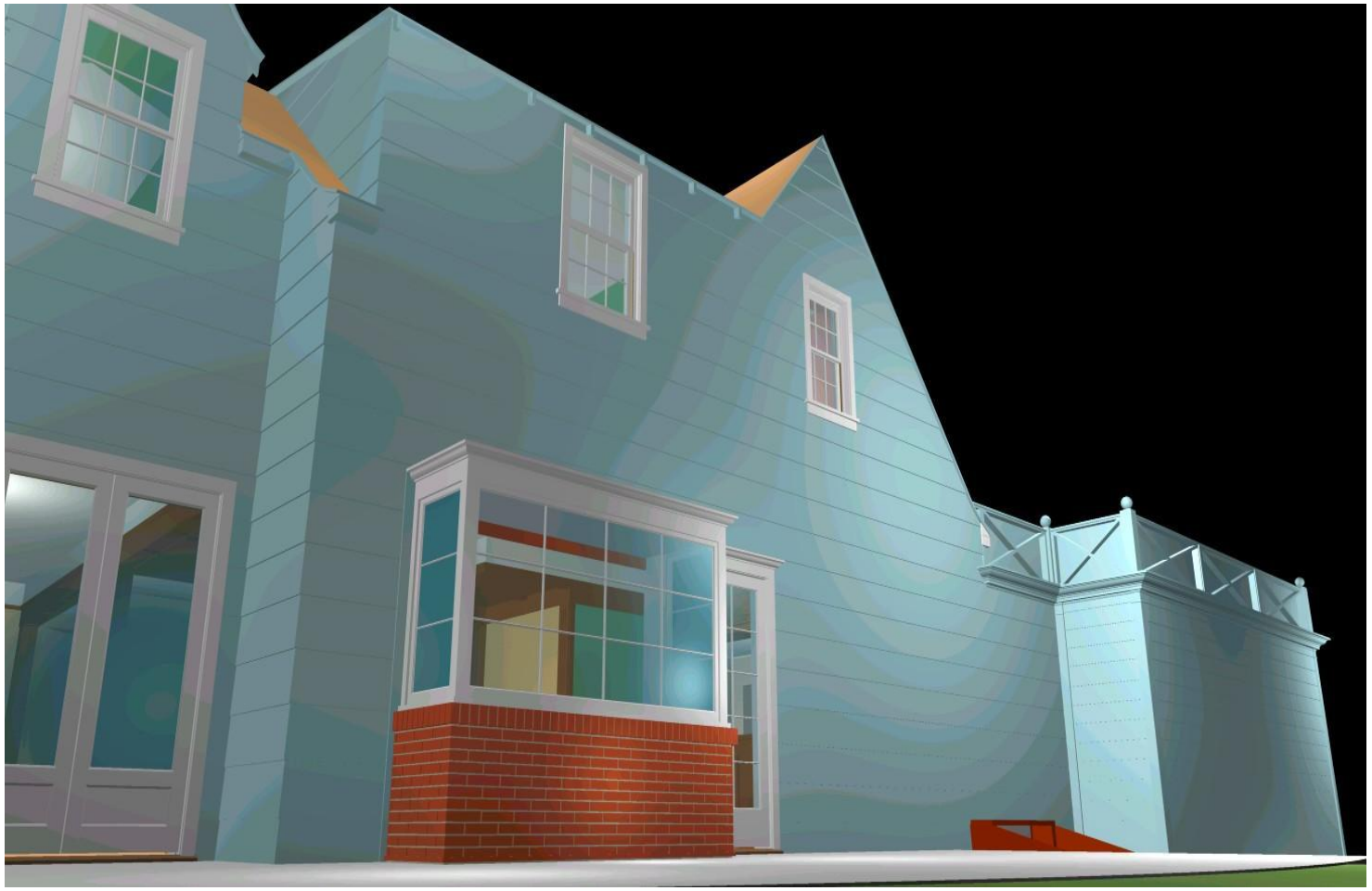
- Методи и алгоритми за математическо представяне на обекти





Геометрично моделиране

Геометрично моделиране



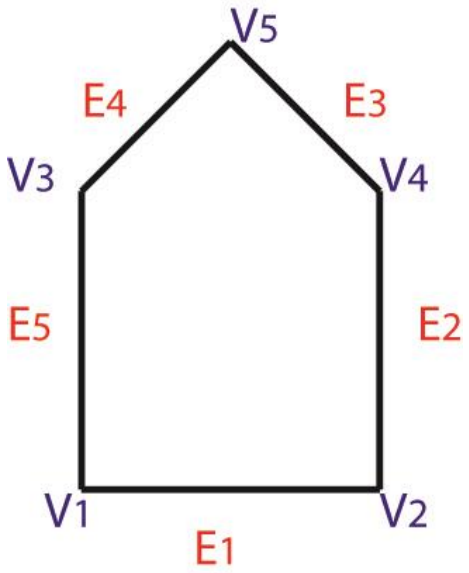
Моделиране на обекти в КГ

- Обекти в КГ
  - съвкупност от полигони
    - наричат се още ***стандартни графични обекти*** или ***примитиви***

- Основни методи за представяне на модели на графични обекти
  - ***равнинни криви и квадратични повърхнини***
    - при визуализиране се свеждат до мрежа от полигони
  - ***сплайнови повърхнини***
  - ***процедурни методи***
    - фрактали и моделиране на поведението на частици
      - particle systems

Представяне на 2D обекти

- ***Прости примитиви***
  - описват се с таблици на възли и дъги
    - всеки възел се описва с координатите си еднократно
    - всяка дъга е подредена двойка от индекси на възли

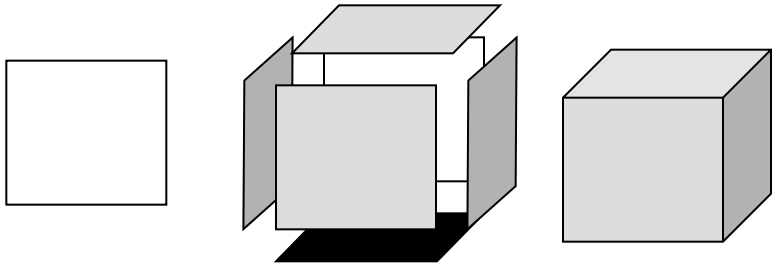


Vertex	
1	(0,0)
2	(1,0)
3	(0,1)
4	(1,1)
5	(0.5,1.5)

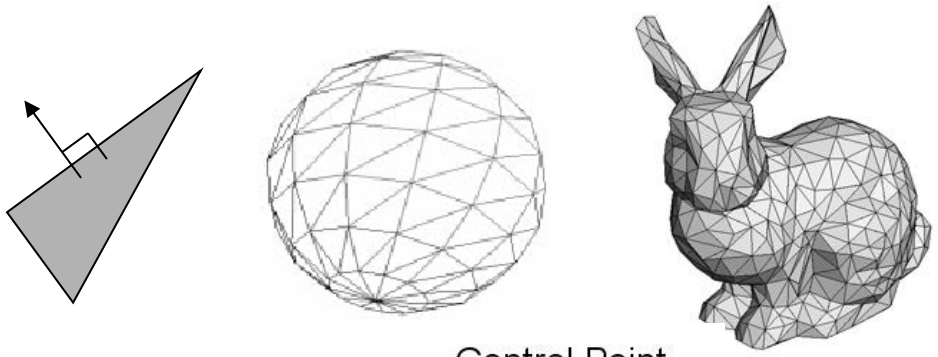
Edge	
1	(1,2)
2	(2,4)
3	(4,5)
4	(5,3)
5	(3,1)

Представяне на 3D обекти

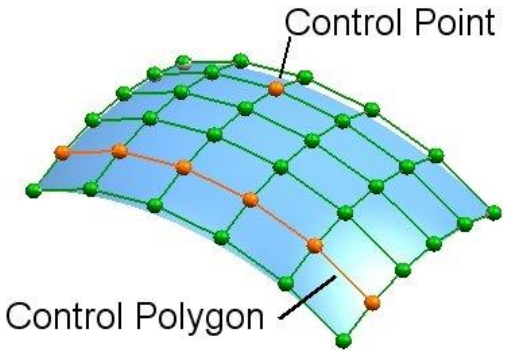
- ***3D примитиви***
  - прости стандартни примитиви



  - мрежа от многоъгълници



  - параметрични повърхнини



Представяне на 3D обекти

- Две основни категории методи
  - ***Представяне на границите на обекта***
    - множество от повърхнини, които отделят вътрешността на обекта от околната среда
  - ***Представяне с разбиване на пространството***
    - пространството, заемано от обекта, се разделя на множество от малки и непокриващи се твърди тела

Моделиране на 3D обекти

- Многостени
- Квадратични повърхнини
- Представяне чрез трансформации
- Конструктивна геометрия на твърди тела

Многостени

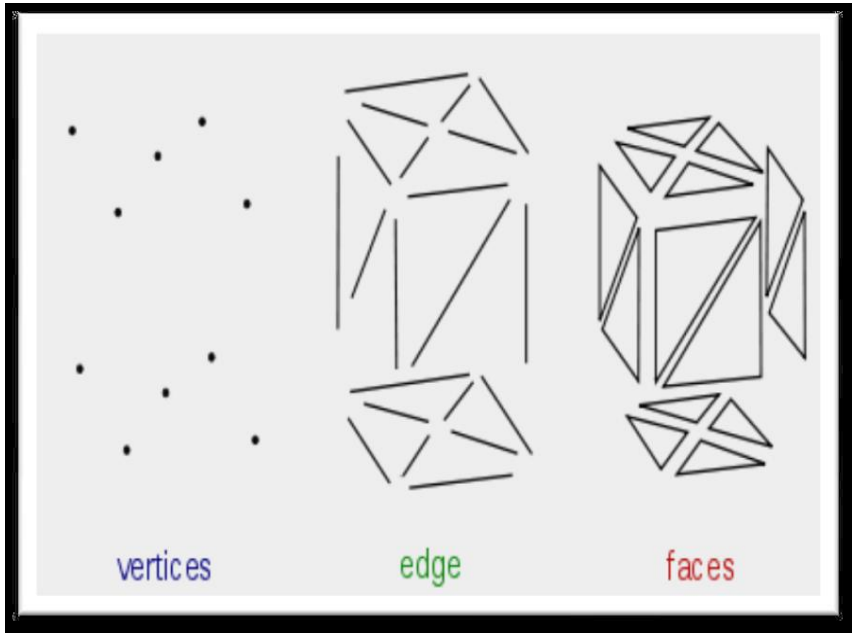
- Обектите се представят чрез множество равнинни полигони, които ограждат вътрешността на обекта
  - най-простият и най-бърз начин за рендиране на обектите
  - могат да се използват и стандартни графични обекти
  - в много случаи приложенията за графично моделиране позволяват да се дефинират обекти като криви повърхности, но в действителност ги конвертират до мрежа от плоски полигони за рендиране
- За да се зададе многостен се определят върховете на изграждащите го полигони

Многостени

- ***Плътни многостени***
  - 3D обектите се описват с множество от стени
  - всяка стена е равнинен многоъгълник (полигон)
    - стандартен графичен обект
- ***Геометрични многостени***
  - призми, пирамиди,...
  - описанието е математически точно
- ***Многостени с други повърхности***
  - апроксимация с мрежа от многоъгълници
    - polygon mesh

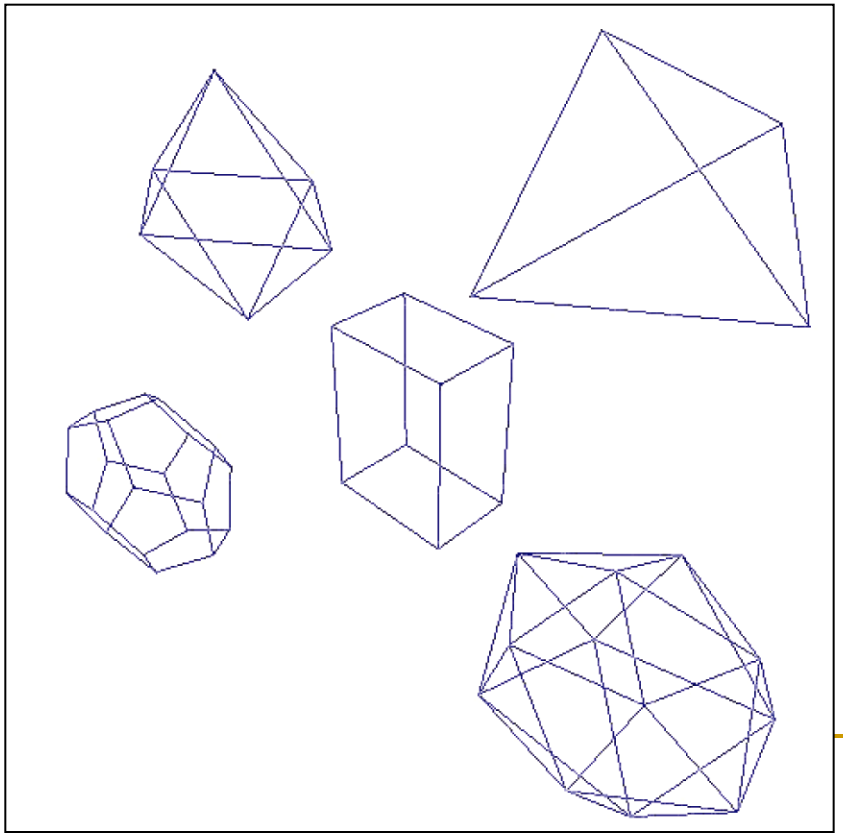
Многостени

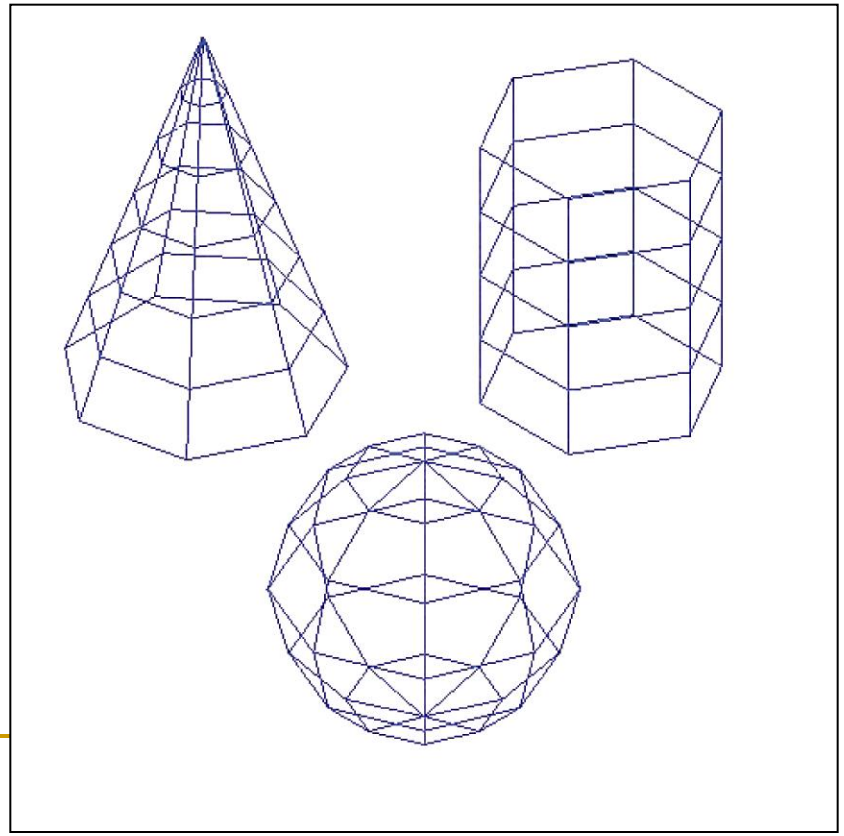
- ***Представяне на мрежа от многостени***
  - възли
  - линии
  - полигони



Многостени

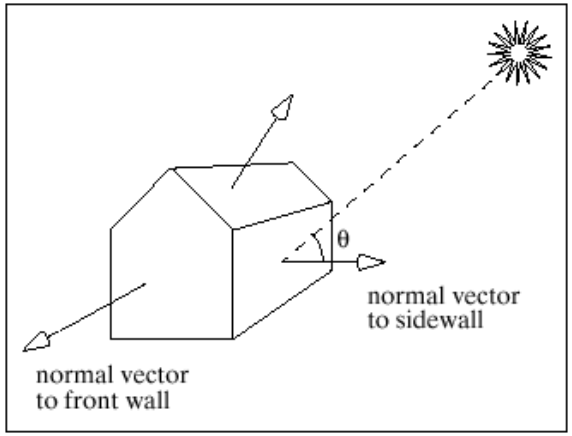
- ***Дефиниране на многостени***
  - задават се с възлите на изграждащите ги





Многостени

- ***Мрежа от многоъгълници***
  - съвкупност от полигони, които представят повърхността на обект
    - полигоните са линейни апроксимации на отделни региони от повърхността
  - обекта се представя с многоъгълници (facet)
  - за всеки многоъгълник се задават
    - възлите на многоъгълника
    - нормала на многоъгълника
      - ориентацията на многоъгълника е важна за осветяване и рендериране



Мрежа от полигони

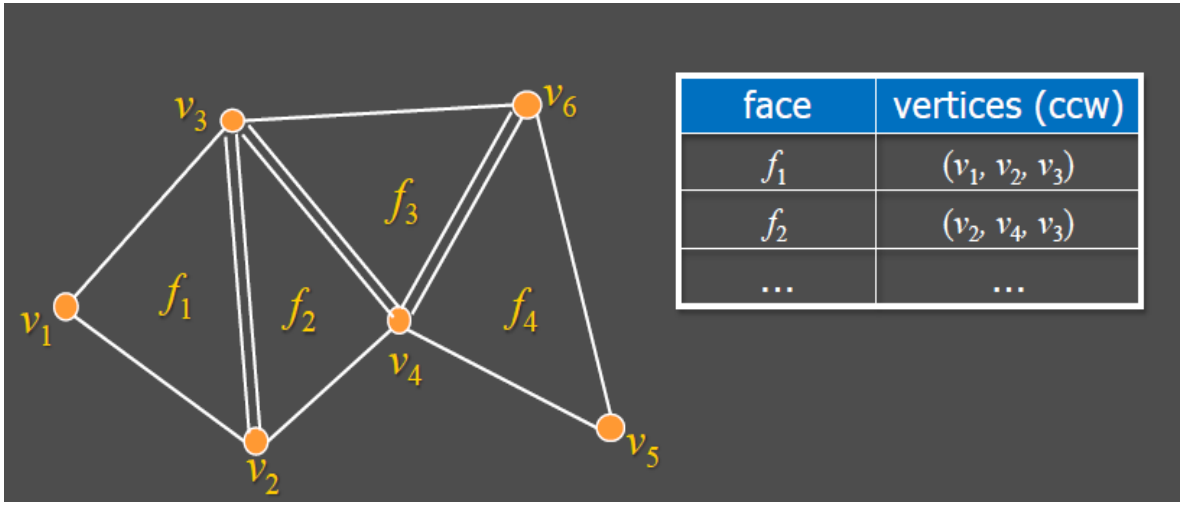
- ***Представяне***

(структури от данни)

  - Polygon Soup
  - Vertex-Vertex
  - Face-Vertex
  - Winged-Edge

Мрежа от полигони

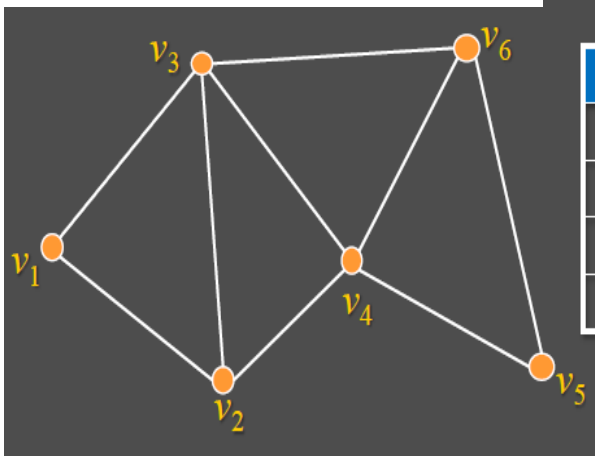
- ***Представяне***
  - ***Polygon Soup***



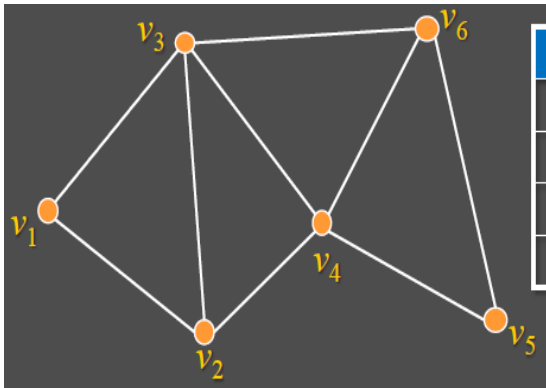
face	vertices (ccw)
f_1	(v_1, v_2, v_3)
f_2	(v_2, v_4, v_3)
…	…

Мрежа от полигони

- ***Представяне***
  - ***Vertex-Vertex***



vertex	coordinates	Neighbors
v_1	(x_1, y_1, z_1)	(v_2, v_3)
v_2	(x_2, y_2, z_2)	(v_1, v_3, v_4)
v_3	(x_3, y_3, z_3)	(v_1, v_2, v_4, v_6)
…	…	

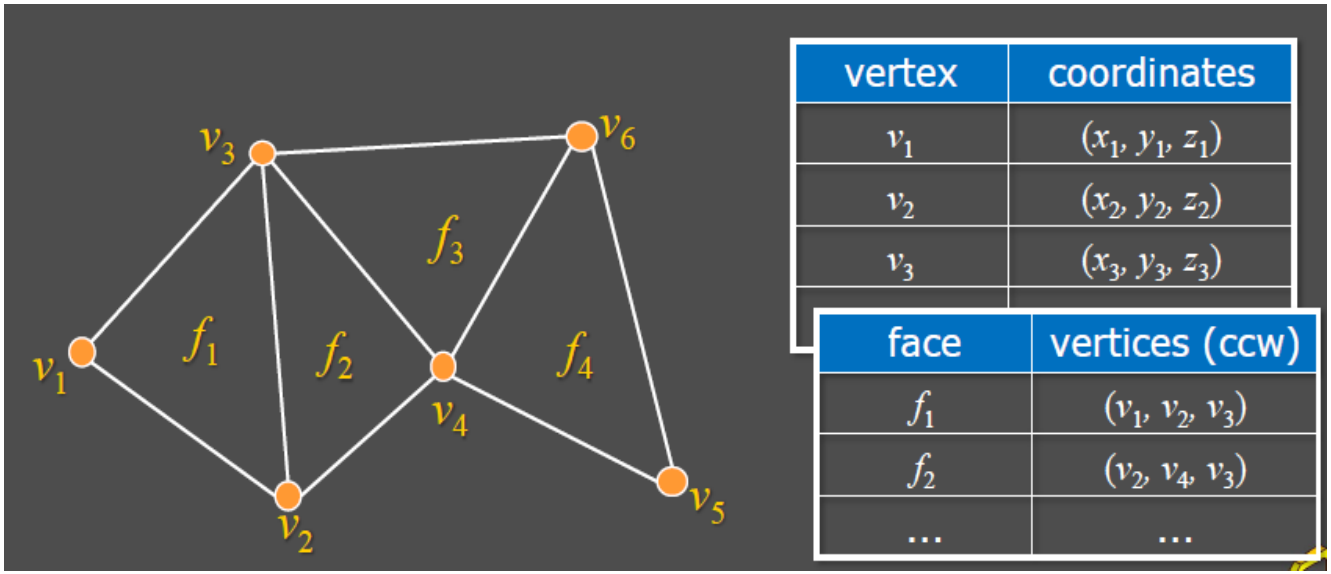


vertex	coordinates
v_1	(x_1, y_1, z_1)
v_2	(x_2, y_2, z_2)
v_3	(x_3, y_3, z_3)
…	

	v_1	v_2	v_3	…
v_1		1	1	
v_2	1		1	
v_3	1	1		
…				

Мрежа от полигони

- ***Представяне***
  - ***Face-Vertex***



vertex	coordinates
v_1	(x_1, y_1, z_1)
v_2	(x_2, y_2, z_2)
v_3	(x_3, y_3, z_3)

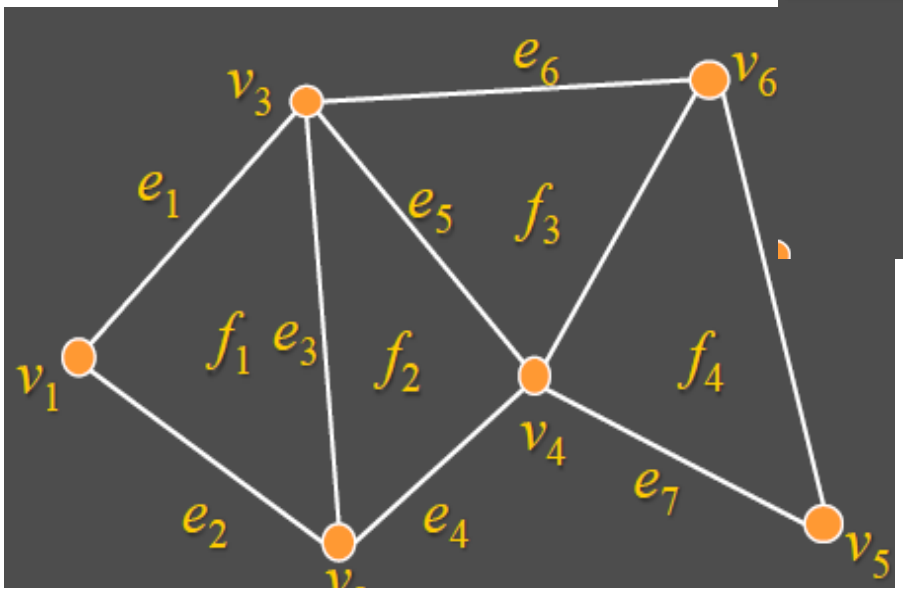
face	vertices (ccw)
f_1	(v_1, v_2, v_3)
f_2	(v_2, v_4, v_3)
…	…

Мрежа от полигони

- ***Представяне***
  - ***Winged-Edge***

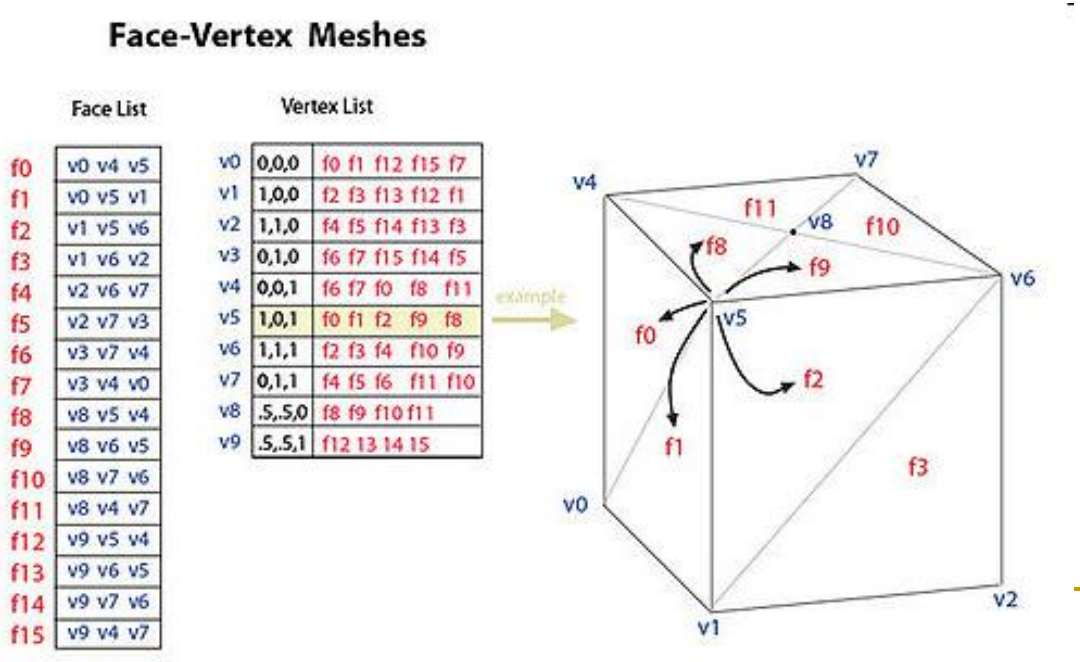
vertex)	coordinates	Edges
v_1	(x_1, y_1, z_1)	(e_1, e_2)
v_2	(x_2, y_2, z_2)	(e_2, e_3)
v_3	(x_3, y_3, z_3)	(e_1, e_5)
…		

face	vertices (ccw)	Edges
f_1	(v_1, v_2, v_3)	(e_1, e_2, e_3)
f_2	(v_2, v_4, v_3)	(v_3, v_4, v_5)
…	…	



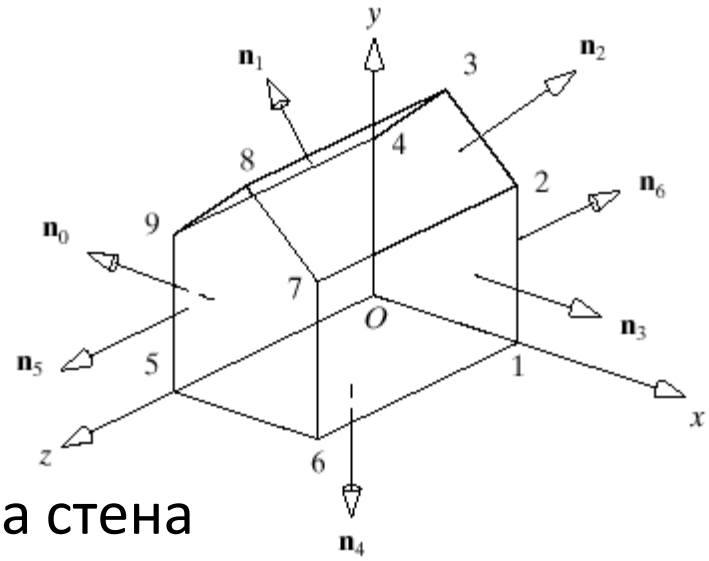
Многостени

- ***Мрежа от триъгълници***
  - разпространено представяне на 3D форми
  - всички възли на даден триъгълник лежат в една равнина
  - лесно се изпълняват различни операции



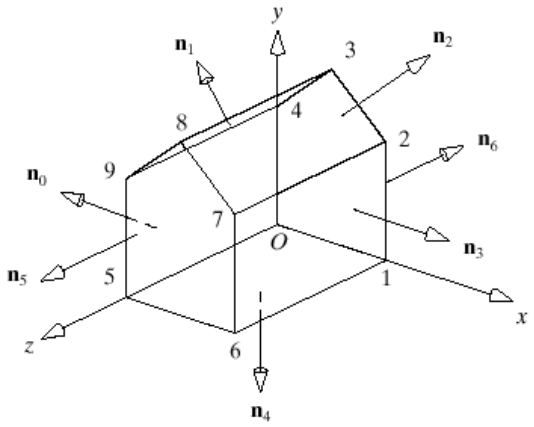
Многостени

- Пример
  - седем многоъгълника
  - 10 възела
  - плоски стени
  - по един нормален вектор за всяка стена



- Структури за представяне на мрежа от многоъгълници
  - съхраняват се уникалните възли и нормалните вектори
  - за всеки многоъгълник се описва множество от индекси на възли и нормални вектори

Многостени



- възли
  - геометрия на обекта
- нормални вектори
  - ориентация
- многоъгълници
  - топология

vertex	x	y	z
0	0	0	0
1	1	0	0
2	1	1	0
3	0.5	1.5	0
4	0	1	0
5	0	0	1
6	1	0	1
7	1	1	1
8	0.5	1.5	1
9	0	1	1

normal	n_x	n_y	n_z
0	−1	0	0
1	−0.707	0.707	0
2	0.707	0.707	0
3	1	0	0
4	0	−1	0
5	0	0	1
6	0	0	−1

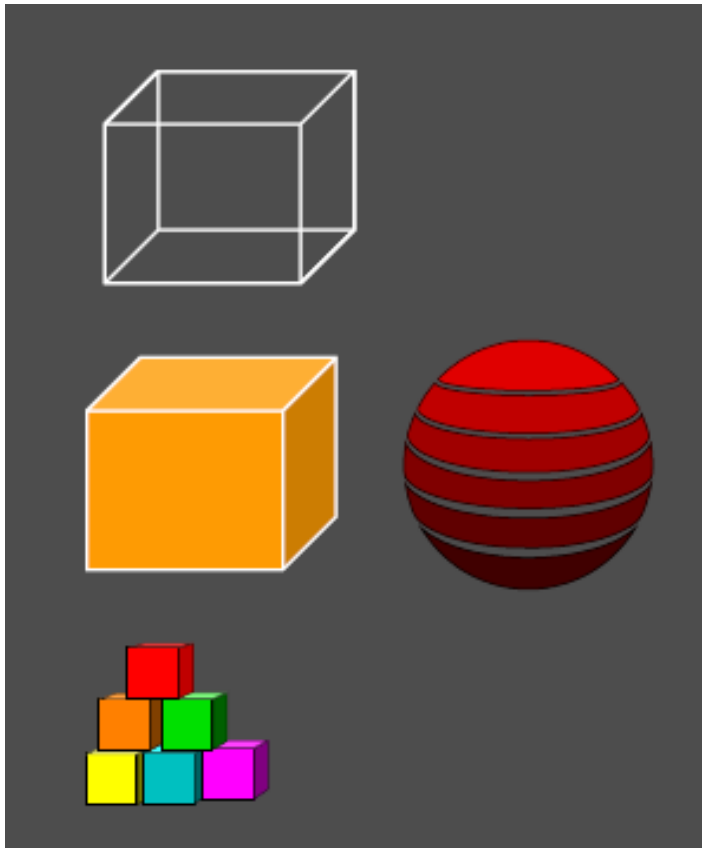
face	vertices	associated normal
0 (left)	0,5,9,4	0,0,0,0
1 (roof left)	3,4,9,8	1,1,1,1
2 (roof right)	2,3,8,7	2,2,2,2
3 (right)	1,2,7,6	3,3,3,3
4 (bottom)	0,1,6,5	4,4,4,4
5 (front)	5,6,7,8,9	5,5,5,5,5
6 (back)	0,4,3,2,1	6,6,6,6,6

Многостени

- ***Мрежа от многоъгълници***
  - характеристики
    - плътност
    - свързаност
    - простота
    - равнинност
    - изпъкналост
  - недостатъци
    - равнинни многоъгълници
    - ограничена разделителна способност
    - трудно се извършват деформации
    - няма естествена параметризация

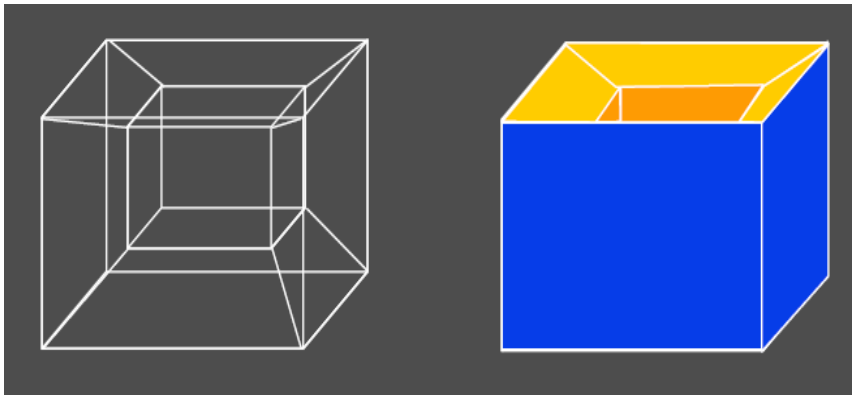
Представяне на многостени

- ***Категории***
  - ***Представяне с контури***
    (Wire-frame)
  - ***Представяне с повърхности***
    (Surface)
  - ***Представяне с обеми***
    (Volumetric)



Представяне с Wire-Frame

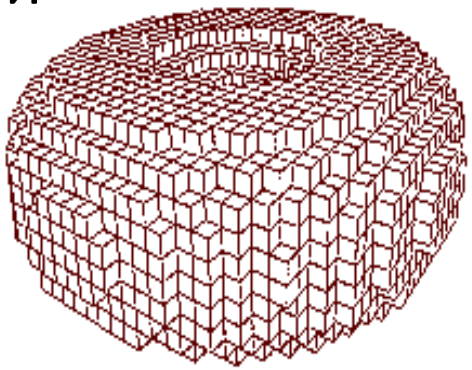
- Представяне на обектите като множество от точки (***node***) и линии (***edge***)
  - граф
  - топологична информация



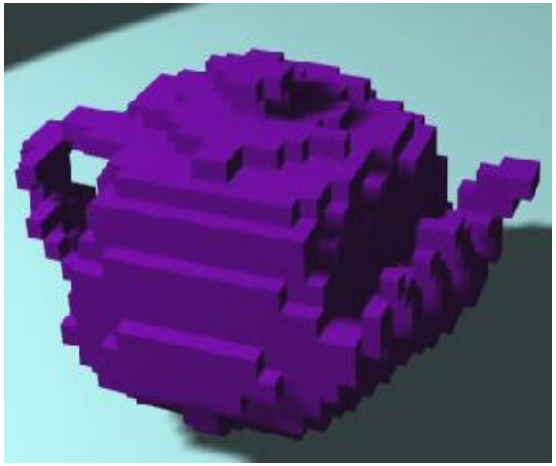
- предимства
  - бързо визуализиране в интерактивни системи
- недостатъци
  - сложно

Представяне с обеми

- Разделяне на пространството на региони
  - ***voxel***



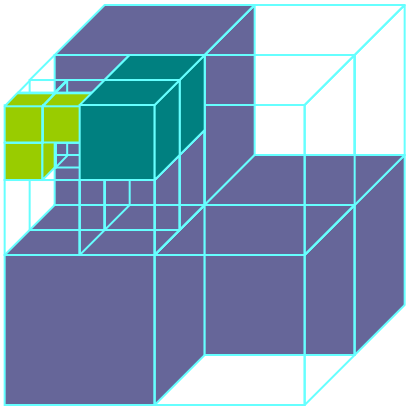
- предимства
  - просто използване на булеви операции
  - просто тестване за принадлежност
  - представяне на вътрешността на обект
- недостатъци
  - неефективно съхранение
  - липсва гладкост
  - трудно се манипулира

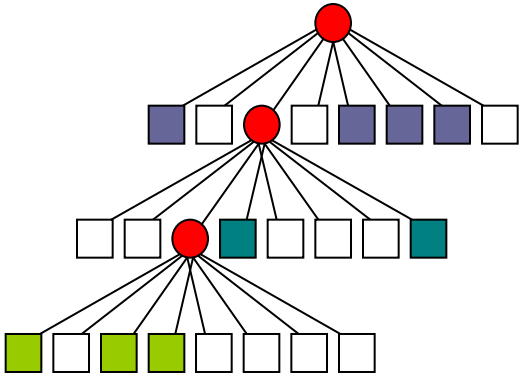


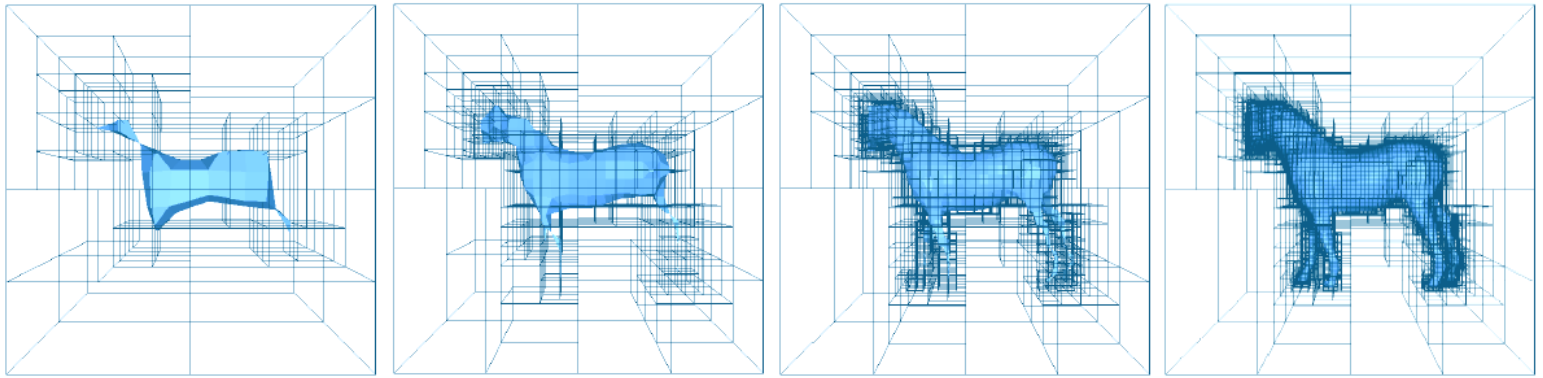
Представяне с осмични дървета

- ***Octree***
  - итеративно разделяне на пространството на кубични под-региони (voxels)
  - условие за край на разделянето
    - еднородност на региона – запълнен или празен
  - предимства
    - удобно съхраняване на 3D модели
    - опростен анализ на скрити стени
    - контрол на степента на детайлност в модела

Представяне с осмични дървета



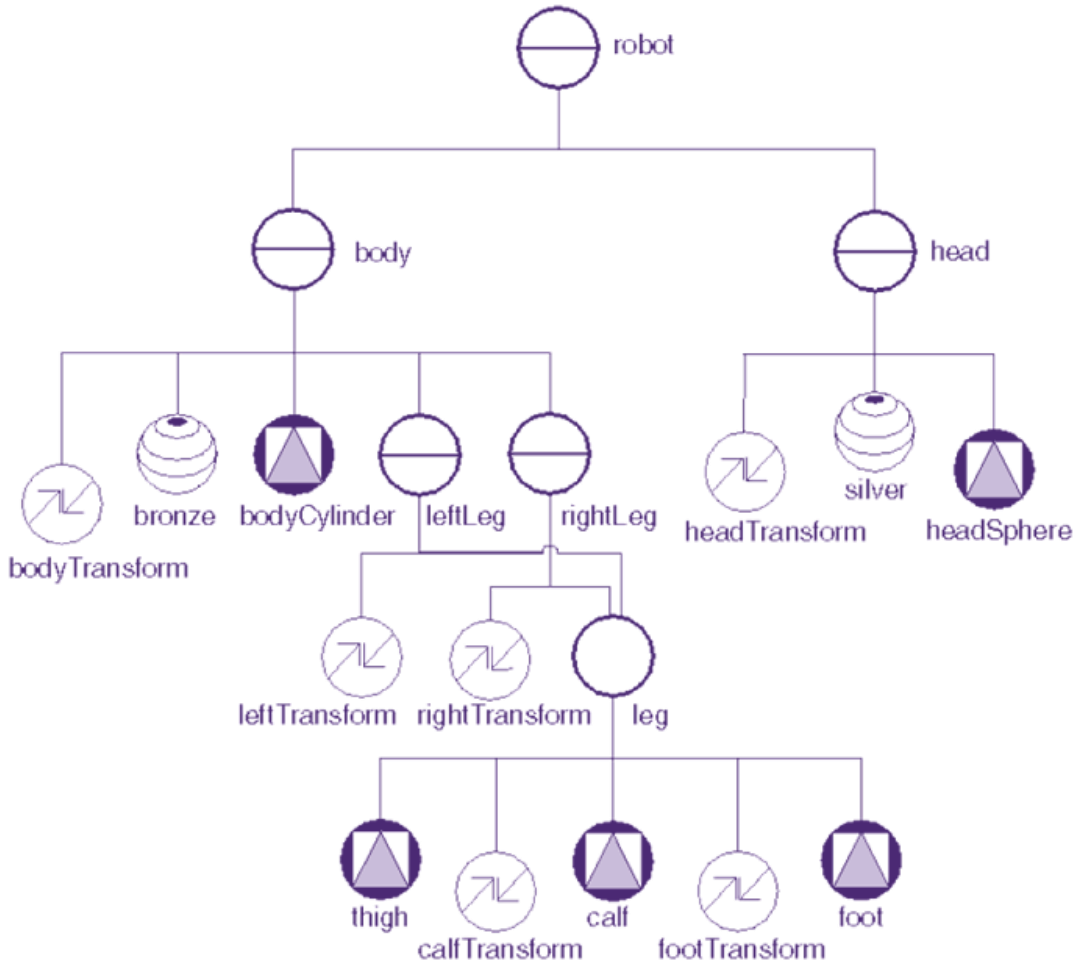


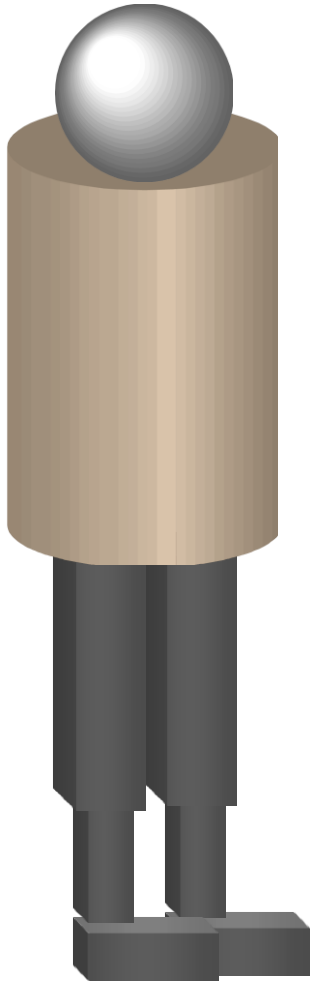


Представяне с граф

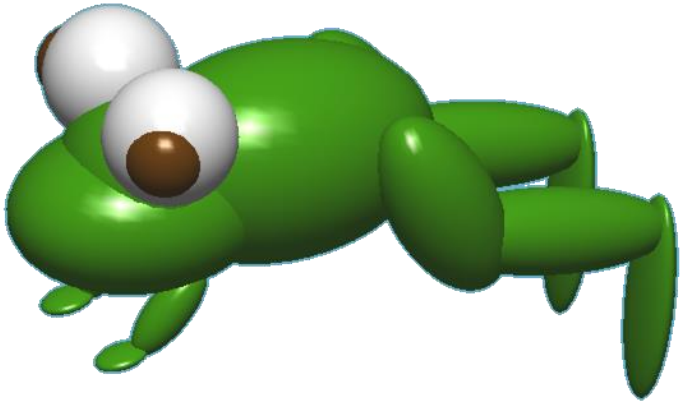
- ***Scene graph***
  - йерархично подреждане на компонентите на графичната сцена
  - предимство
    - моделиране и визуализиране на вторичните компоненти се опростява заради наследяването на основните компоненти

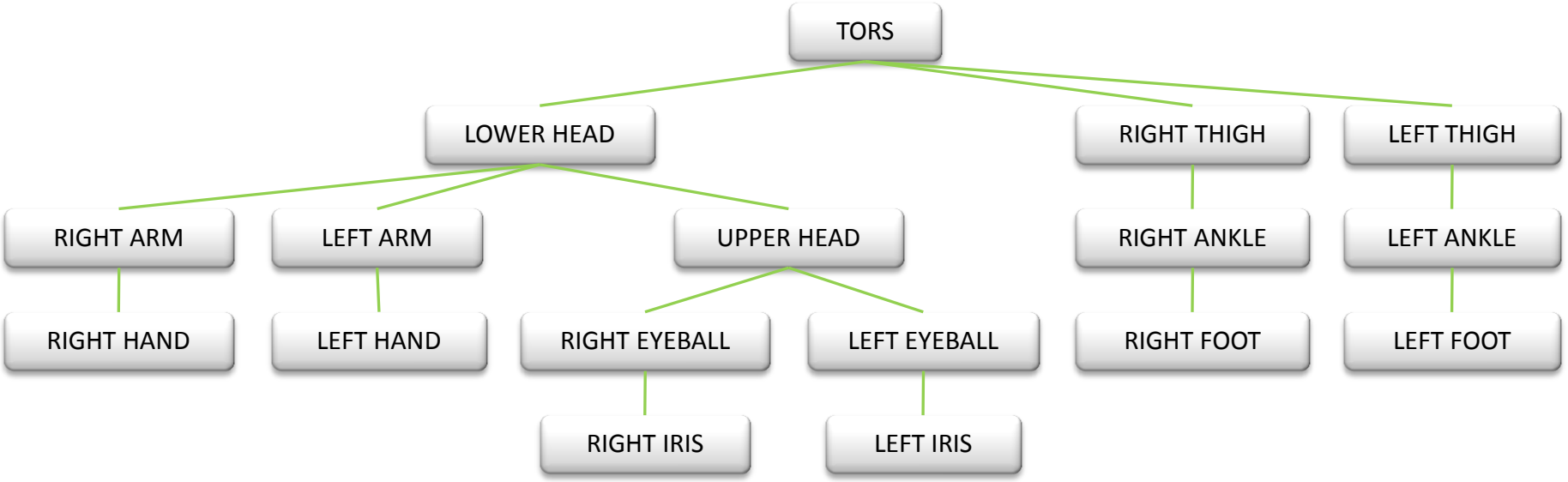
Представяне с граф





Представяне с граф





Моделиране на 3D обекти

- Многостени
- ***Квадратични повърхнини***
- Представяне чрез трансформации
- Конструктивна геометрия на твърди тела

Квадратични повърхности

- Често използван подход за моделиране на класове обекти
  - 3D повърхнините се описват ***с квадратични уравнения***
- Квадратични повърхности
  - сфера
  - елипсоид
  - тороид
  - параболоид
  - хиперболоид

Сфера

- Сферична повърхност с радиус r и център в началото на координатната система
  - множеството от точки (x, y, z), за които е изпълнено

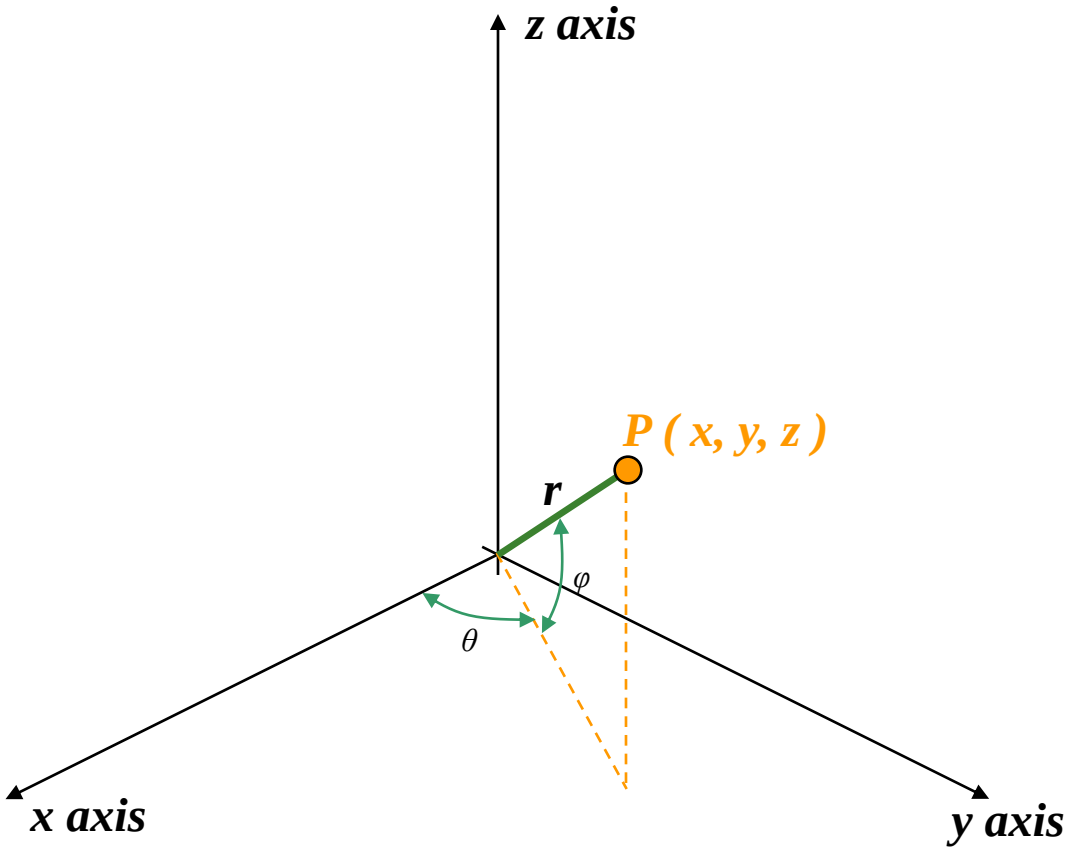
$$x^2 + y^2 + z^2 = r^2$$

- Параметрично представяне

$$x = r\cos\phi\cos\theta$$
$$y = r\cos\phi\sin\theta$$
$$z = r\sin\phi$$

$$-\pi/2 \le \phi \le \pi/2$$
$$-\pi \le \theta \le \pi$$

Сфера



Сфера

- Представяне на сфера в OpenGL с използване на GLUT

```
GLUquadricObj *mySphere

mySphere=gluNewQuadric();
//create the new sphere object

gluQuadricDrawStyle(mySphere,GLU_FILL);
// some other styles: GLU_POINT, GLU_LINE

gluSphere(mySphere,1.0,12,12);
// radius, # longitude lines, # latitude lines
```

Квадратични повърхности

- Представяне на квадратични повърхности в OpenGL с използване на GLUT
  - предефинирани обекти с квадратични повърхности
  - `glutWire***()`
  - `glutSolid***()`
    - примери
      - `glutWireCube(size); glutSolidCube(size);`
      - `glutWireSphere(radius,nlongitudes,nlatitudes);`
      - `glutWireCone(rbase,height,nlongitudes,nlatitudes);`
      - `glutWireTeapot(size);`
      - и др.

Моделиране на 3D обекти

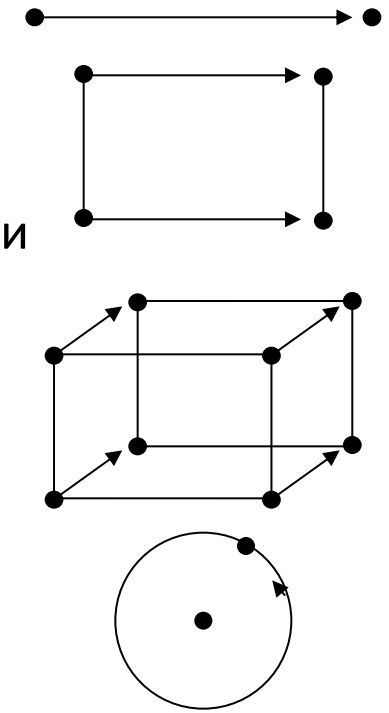
- Многостени
- Квадратични повърхнини
- ***Представяне чрез трансформации***
- Конструктивна геометрия на твърди тела

Представяне чрез трансформации

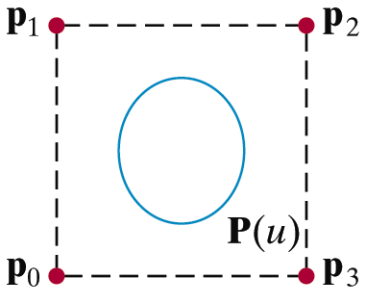
- ***Sweep presentation***
  - полезно представяне на 3D обекти с транслационна, ротационна и други симетрии
  - обектите се задават като двумерни модели и преобразуване, което премества тези модели в пространството

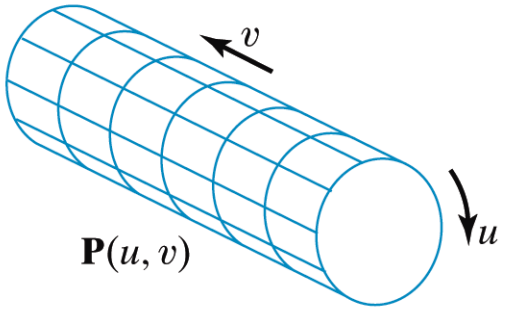
Представяне чрез трансформации

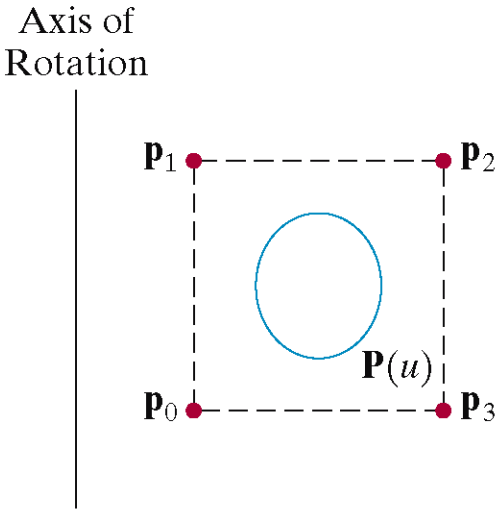
- ***Generative object description***
  - представяне на 3D чрез движение на 2D (възли + движение)
- Линия
  - път на движение на точка (едномерен случай)
- Квадрат
  - път на движение на линия перпендикулярно на себе си (двумерен случай)
- Куб
  - път на движение на възлите на квадрат перпендикулярно на себе си (тримерен случай)
- Окръжност
  - път на движение на точка при завъртането ѝ на фиксирано разстояние около център

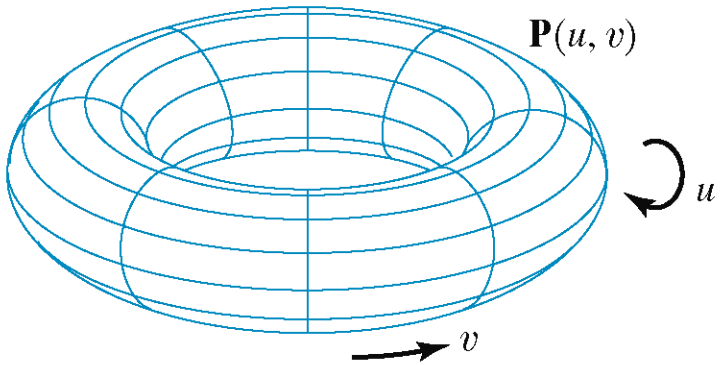


Представяне чрез трансформации









Моделиране на 3D обекти

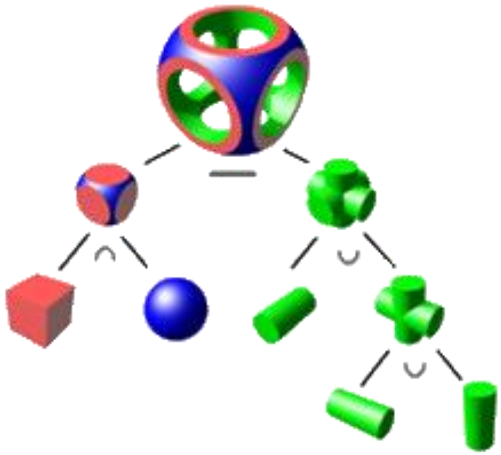
- Многостени
- Квадратични повърхнини
- Представяне чрез трансформации
- ***Конструктивна геометрия на твърди тела***

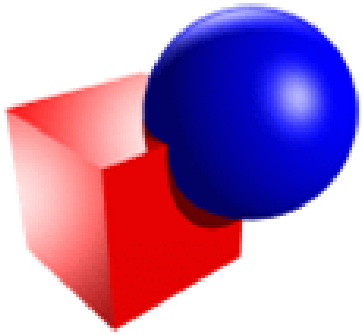
Констр. геометрия на твърди тела

- ***Йерархични модели***
  - частите на модела зависят един от друг
- ***Constructive Solid Geometry (CSG)***
  - методи за представяне на обекти чрез плътни геометрични многостени и теоретико-множествени операции между тях
- Използвани теоретико-множествени операции
  - обединение
  - сечение
  - разлика

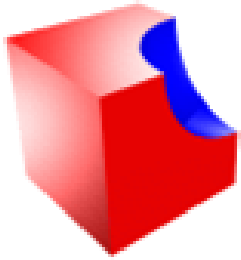
Конструктивна геометрия

- ***Constructive Solid Geometry***
  - метод за моделиране на 3D обекти чрез комбиниране на примитивни геометрични форми с булеви операции





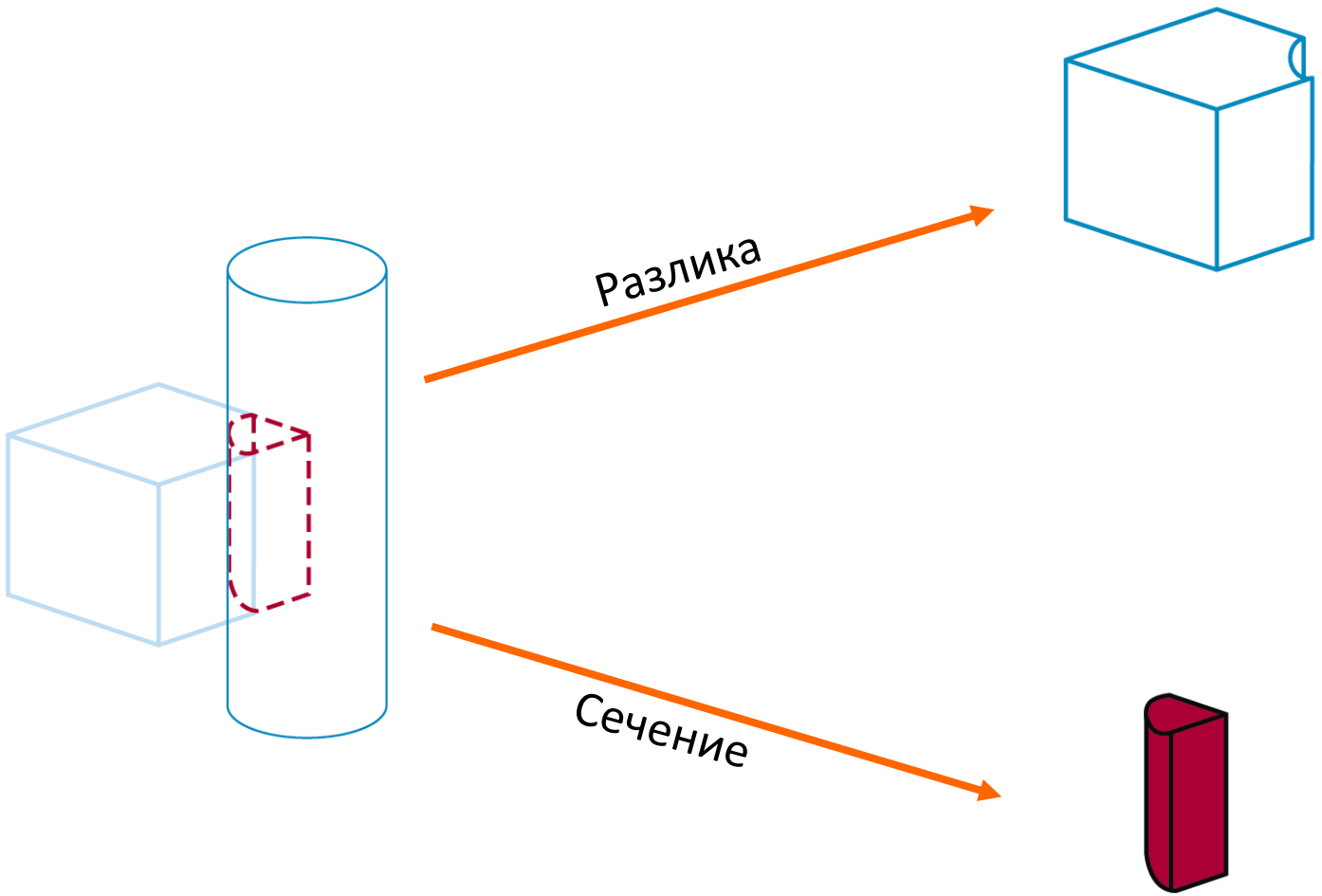
обединение



разлика

сечение

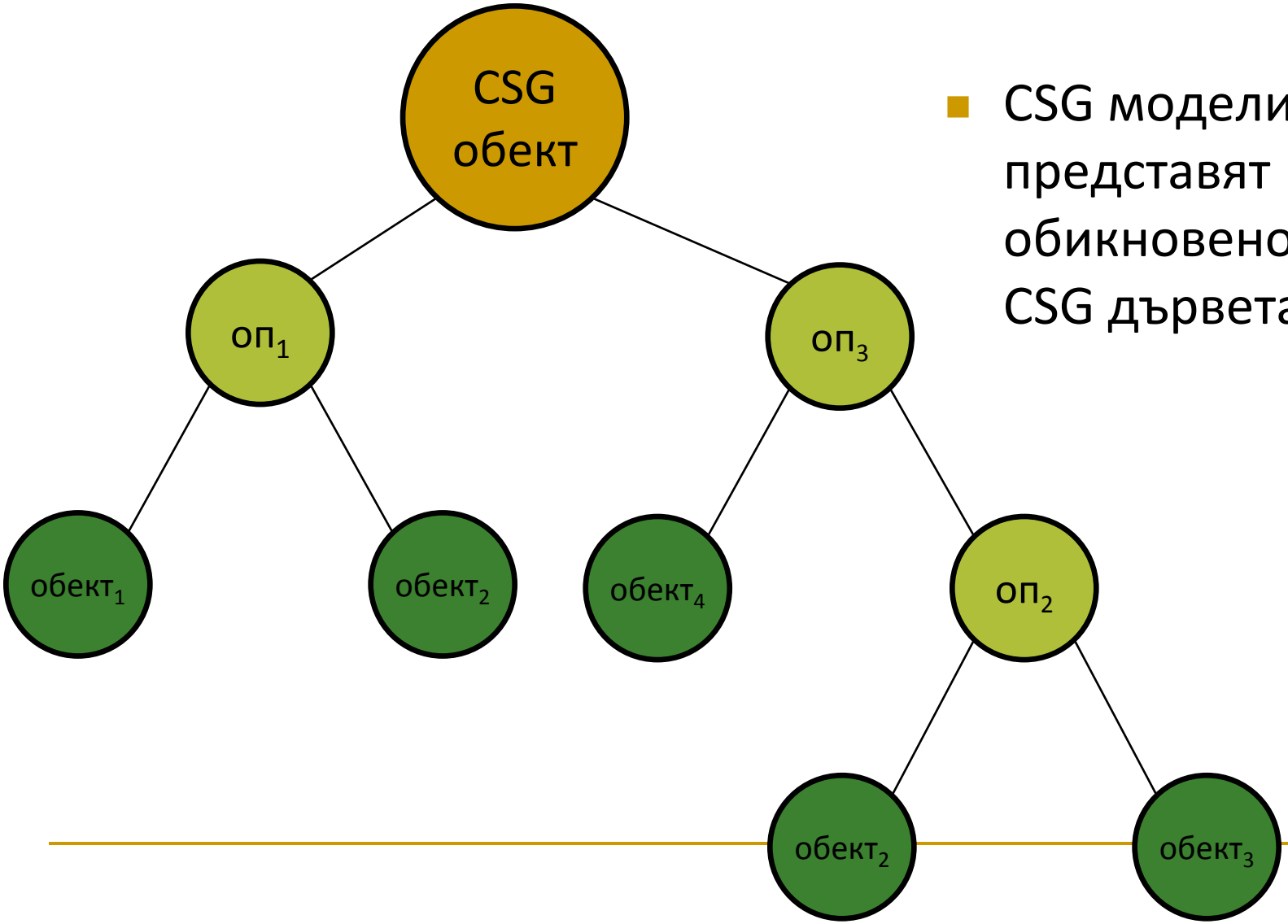
Констр. геометрия на твърди тела



Констр. геометрия на твърди тела

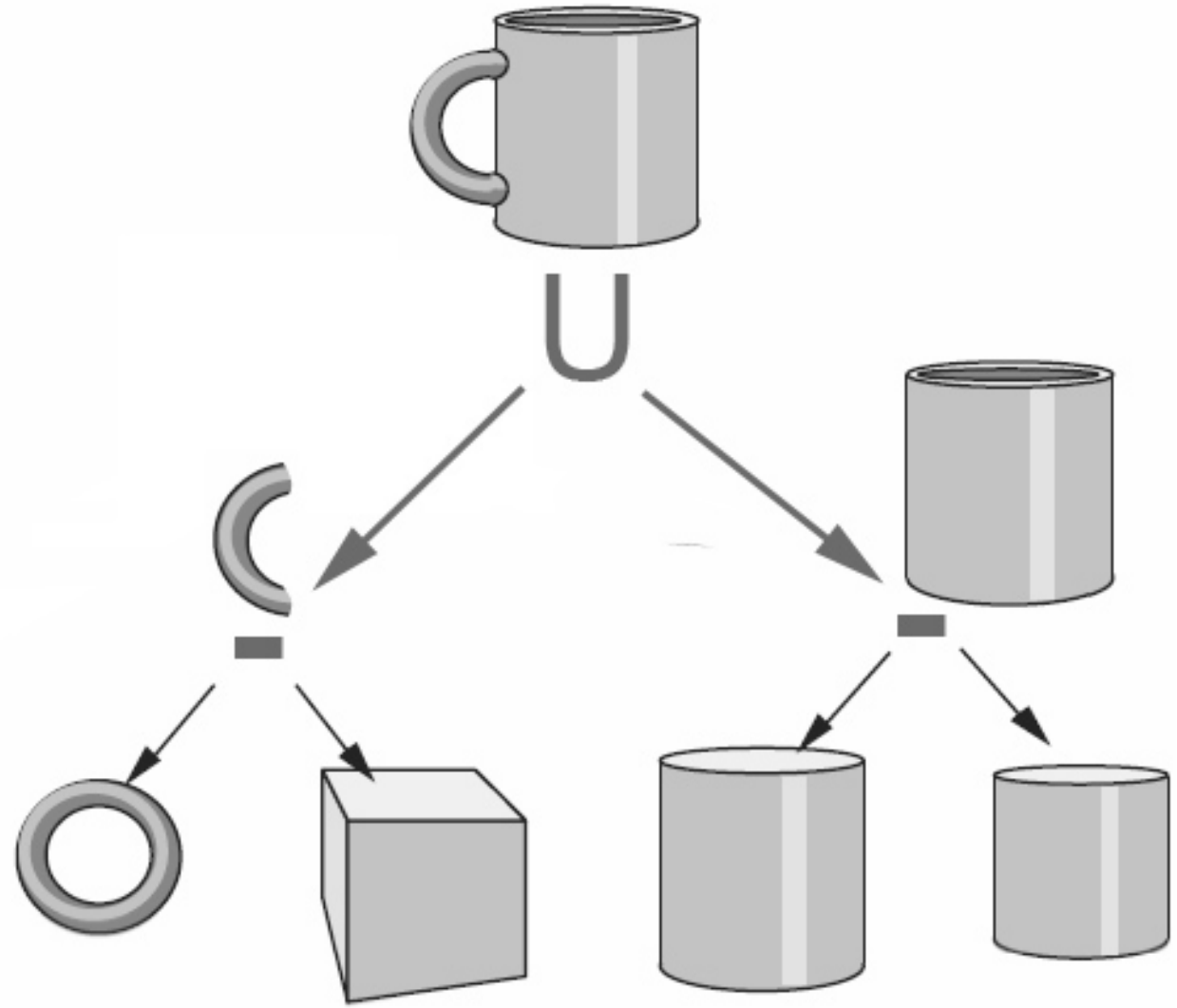
- Използва се малко можество примитиви
  - куб, пирамида, сфера, конус
- Модел на нов обект се създава от моделите на два обекта, комбинирани чрез теоретико-множетсвена операция
- Новите модели могат да се използват за създаване на друг модел
- Процесът продължава докато се моделира цялата сцена с всички необходими обекти

Констр. геометрия на твърди тела

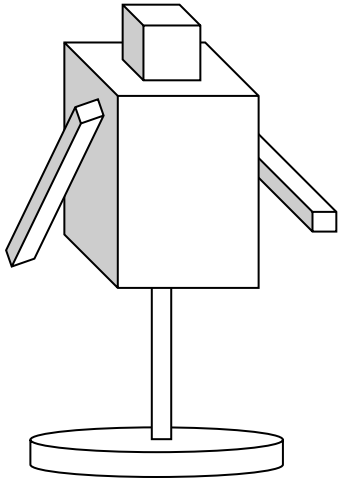


- CSG моделите се представят обикновено като CSG дървета

Констр. геометрия на твърди тела

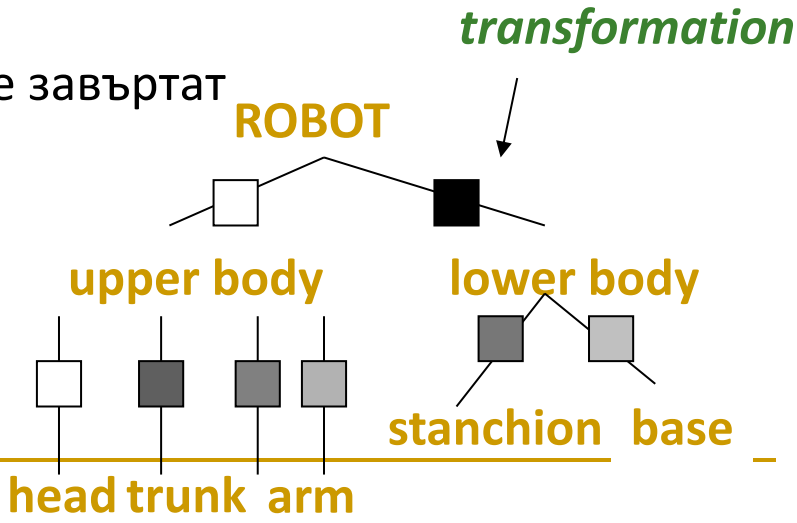


Констр. геометрия на твърди тела



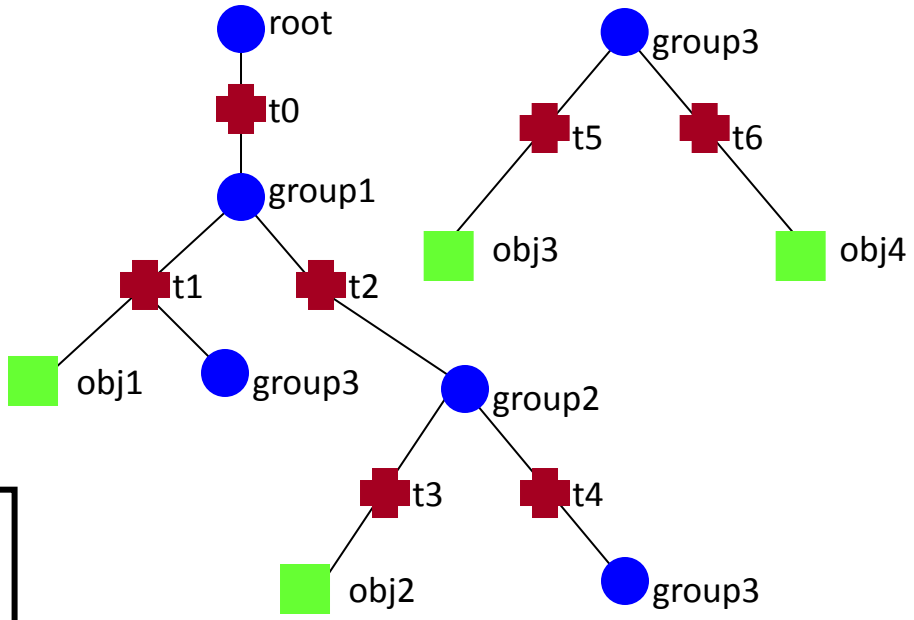
- Трансформациите на възел родител се прилагат и за възлите-деца

- ***Пример***: робот
- онова, долна част , горна част
  - роботът се завърта
    - долната и горната част също се завъртат
  - горната част се завърта
    - главата и рамото също се завъртат

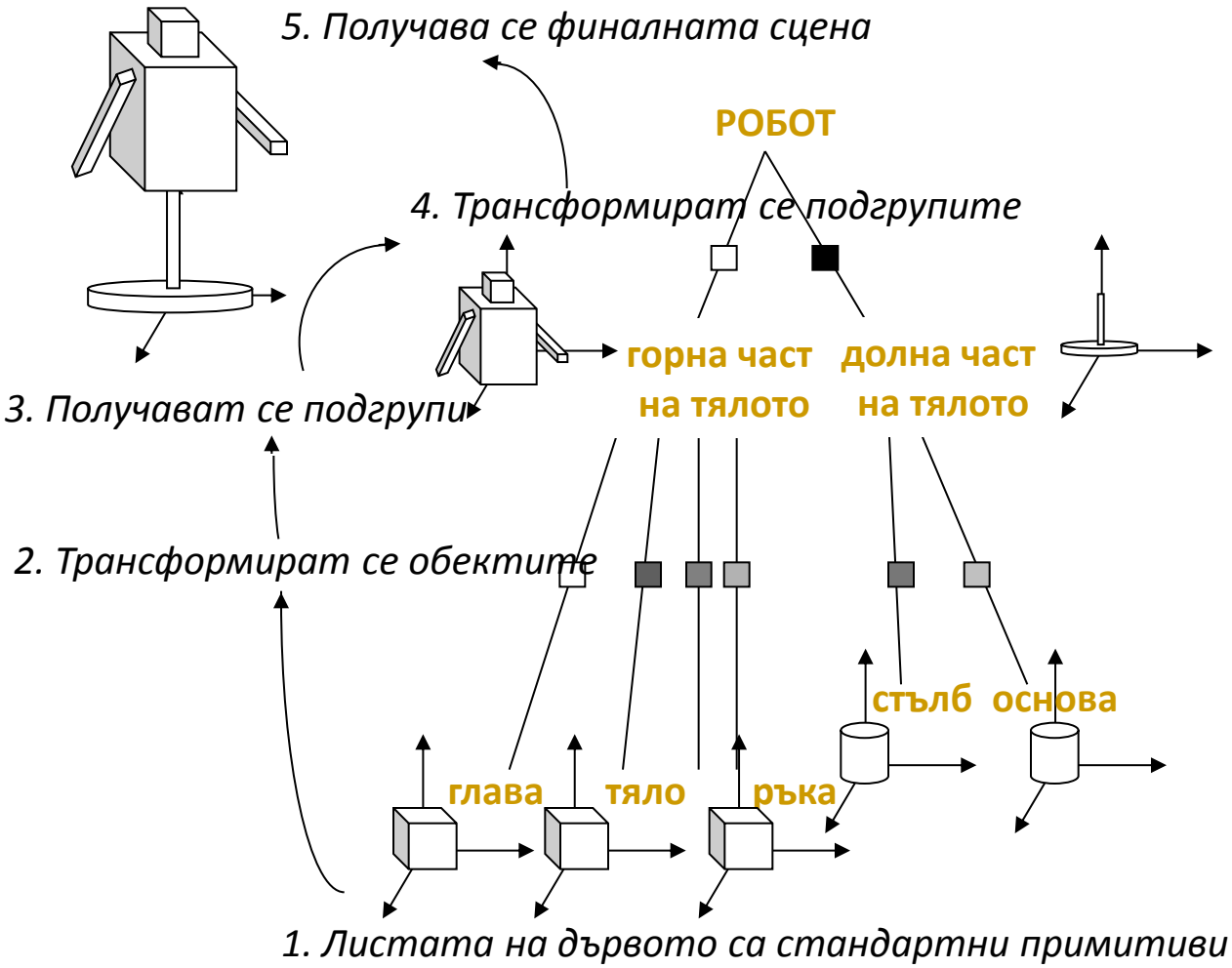


Констр. геометрия на твърди тела

- Трансформациите променят всички възли-деца
- Под-дърветата се наричат подгрупи и могат да се използват отново като обекти
- Обект може да се подложи на различни трансформации
  - например възел group3 се използва 2 пъти
    - след трансформация t1
    - след трансформация t4



Констр. геометрия на твърди тела

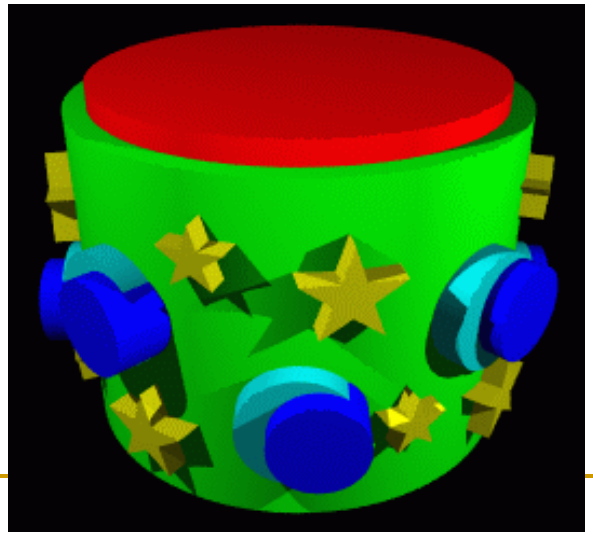


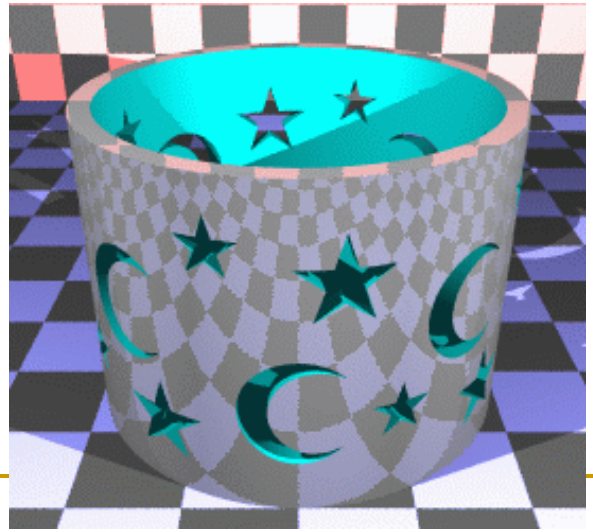
Констр. геометрия на твърди тела

- Съхраняване на 3D сцени
  - Граф на сцената (***Scene Graph***)
    - Directed acyclic graph (DAG)
- Типичен формат на граф на сцената
  - обекти
    - куб, сфера, конус, многостен, и т.н.
  - съхраняват се чрез възлите си
  - задават се атрибутите им
    - цвят, текстура, и др.
  - трансформации
    - възли в графа на сцената

Конструктивна геометрия

- ***Предимства***
  - просто моделиране
  - херархичност
- често прилаган подход в CAD/CAM системите





Констр. геометрия на твърди тела

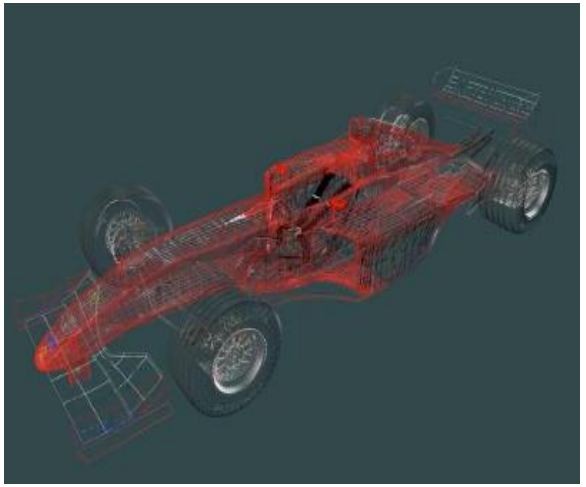
- ***Йерархични модели в OpenGL***
  - имплементират се с използване на матричен стек
    - матричният стек съхранява матриците за проектиране и трансформация на моделите на обектите и за визуализиране на обектите (projection & model-view)
  - използват се функциите за промяна съдържанието на матричния стек
    - `glPushMatrix();`
    - `glPopMatrix();`
- Определя се визуализиране на цял обект по зададен модел, като се запазва модела за генериране и на друг модели
  - с използване на геометрични трансформации

Констр. геометрия на твърди тела

- ***Йерархични модели в OpenGL***
- Създаване на йерархично представяне на сцена (дърво)
  - моделът на всеки обект се представя в собствена координатна система
  - обхожда се дървото и се прилагат трансформациите за преобразуване на моделите на обектите в световна координатна система
  - *правило за обхождане*
    - при преход наляво от възел към възел с необходен десен възел-наследник се изпълнява glPushMatrix();
    - при връщане обратно в този възел се изпълнява glPopMatrix();

Моделиране на сложни форми

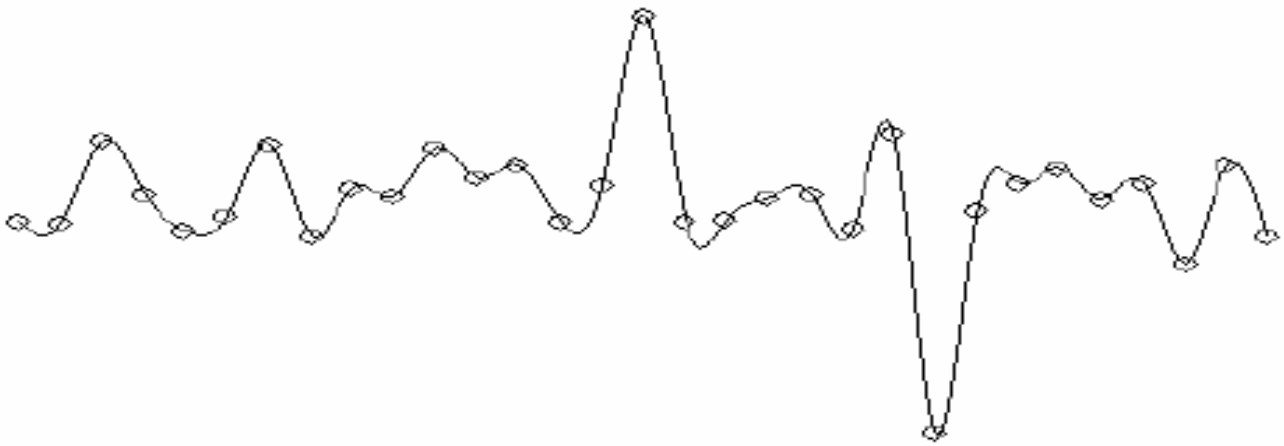
- Апроксимиране с прости примитиви
  - линии, полигони



- При сложни обекти
  - недостатъци
    - голям брой примитиви
    - може да се използва уравнение на сфера, но не и на телефон или човешко лице
  - решение
    - по-сложни примитиви
      - криви (2D) и повърхнини (3D)

Моделиране на сложни форми

- Изисквания към криви/повърхнини в КГ
  - локално управление на формата
    - за да се създава и модифицира лесно
  - гладкост, непрекъснатост
  - възможност за оценяване на производни
  - лесно визуализиране

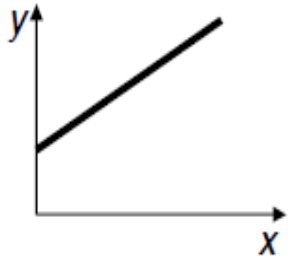


Представяне на криви

- Три форми за представяне на криви в пространството
  - ***явно***
  - ***неявно***
  - ***параметрично***

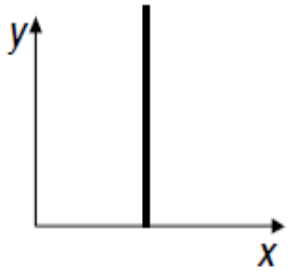
Представяне на криви

- ***Явно***
  - y = f(x)
    - y = mx + b
  - лесно се генерират точки
  - трябва да бъде функция
    - ограничение
      - вертикални линии

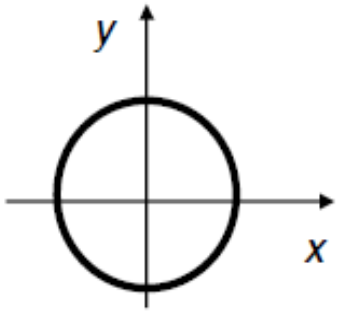


Представяне на криви

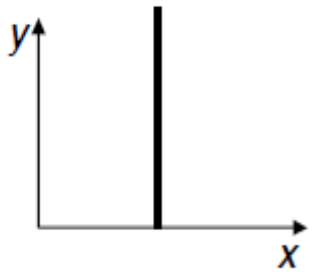
- Явно: y = f(x)



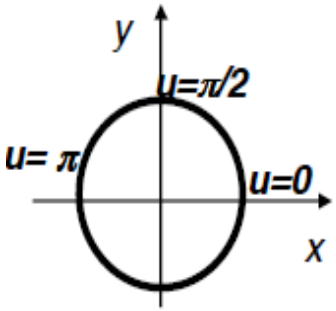
- ***Неявно***
  - f(x,y) = 0
    - $x^2 + y^2 – r^2 = 0$
  - лесно се проверява дали точка лежи на кривата
  - трудно се генерират точки



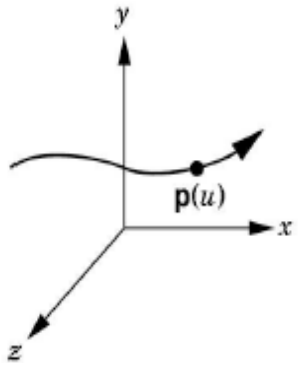
Представяне на криви



- Явно: y = f(x)
- Неявно: f(x,y) = 0

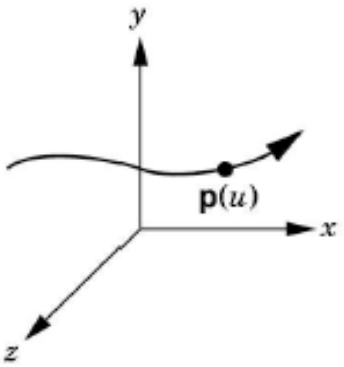


- ***Параметрично***
  - (x,y) = (f(u), g(u))
    - (x, y) = (cos u, sin u)
  - лесно се генерират точки



Параметрично представяне

- ***Параметризиране на крива***
  - зависимост между промените на *u* и промяната на вида на дадена крива в xyz пространството



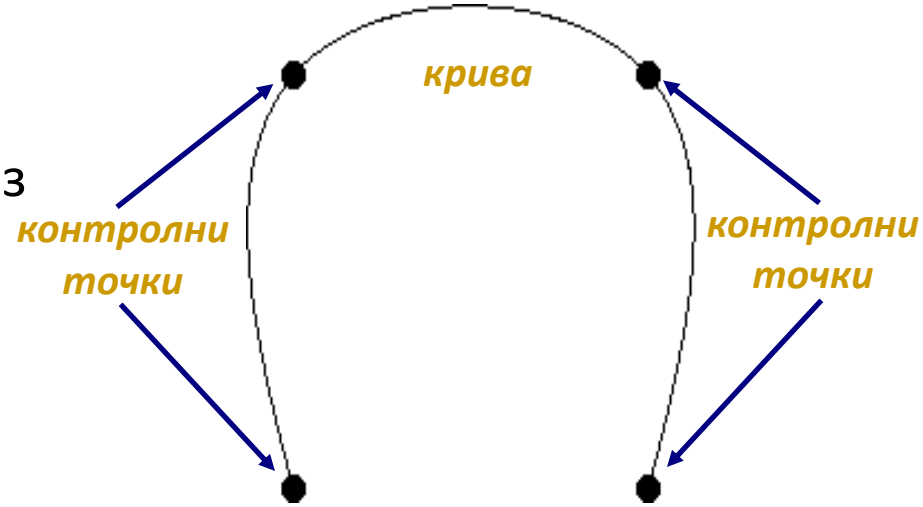
- Защо криви, а не полилинии?
  - редуцира се броя точки
  - лесно се имплементират интерактивни промени
- Защо параметрични, а не криви от вида y,z=f(x)?
  - лесно се представят произволни криви
  - ротационна инвариантност
- Защо параметрични, а не явни?
  - по-просто
  - по-ефективно

Параметрични криви

- ***Параметрични криви***
  - аналогично на траектория на обект в пространството
  - параметър t
    - аналогичен на време на движение
  - чрез параметъра се задава
    - позиция
    - тангентна скорост
    - кривина

Сплайн криви

- ***Параметрични криви***
  - гладка крива минаваща през зададени контролни точки



Pierre Bézier

- ***криви на Безие***
  - метод за апроксимация на сплайнови криви
  - предложен от френския инженер Пиер Безие за дизайн на корпуса на автомобилите на Рено

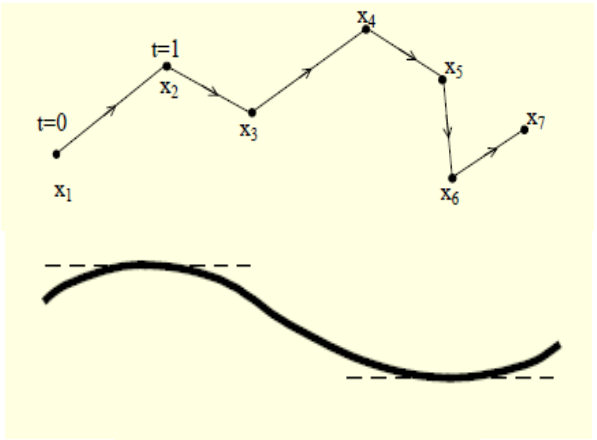
Сплайн криви

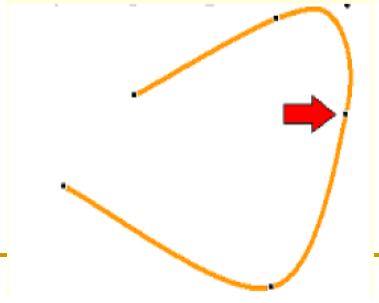
- Крива със свободна форма минаваща през или в близост до множество точки

- Стандартни входни данни
  - множество контролни точки $\{P_i\}$ $i = 0, n$

Сплайн криви

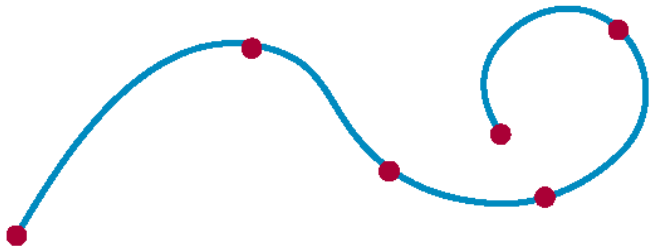
- Параметрични криви, при които линейната интерполация с прави линии се обобщава за интерполационни параметри от по-висок ред
  - линейна интерполация
  - квадратична интерполация
  - кубична интерполация



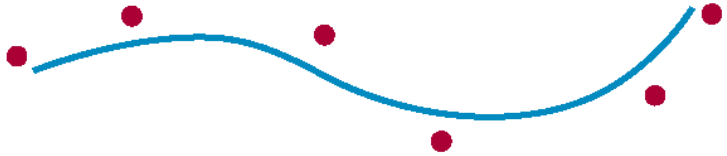


Сплайн криви

- Два подхода за определяне на крива според зададени контролни точки
  - ***Интерполация***
    - кривата минава през всички контролни точки

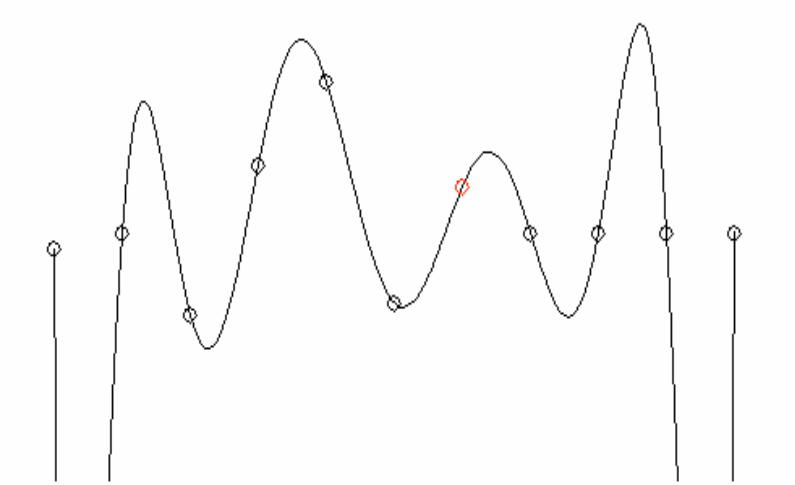


  - ***Апроксимация***
    - кривата не минава през всички контролни точки



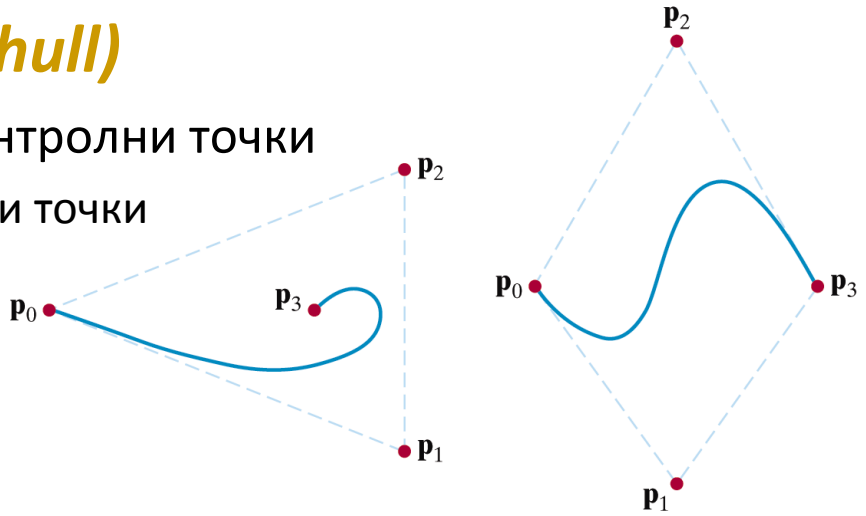
Сплайн криви

- ***Полиномна интерполация***
  - кривата се апроксимира с *n* степенен полином в *n+1* точки
    - нарича се интерполация на Лагранж
  - голямо изместване между контролните точки
  - обикновено се изисква получаване на колкото е възможно по-гладки криви
    - методите с полиноми с голяма степен дават лоши резултати

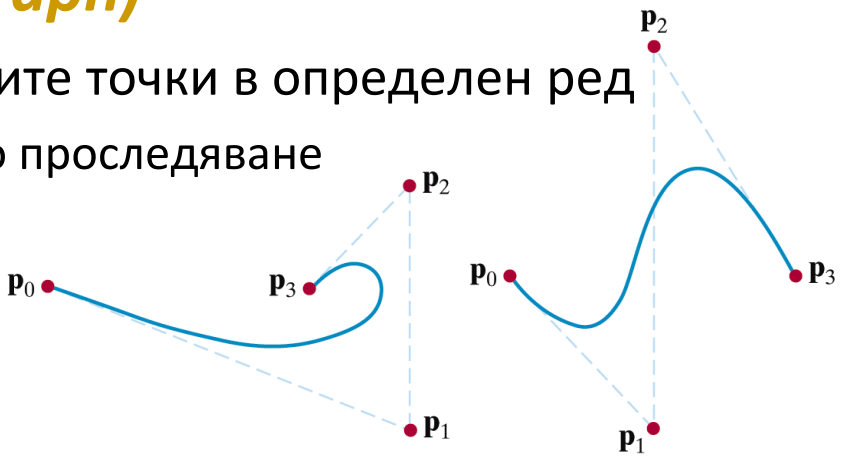


Сплайн криви

- ***Изпъкнала обвивка (convex hull)***
  - граница, определена от всички контролни точки
    - ластик обхващащ всички контролни точки



- ***Управляващ граф (control graph)***
  - полилиния, свързваща контролните точки в определен ред
    - обикновено се използва за лесно проследяване на сплайновите криви



Сплайн криви на Безие

- Сплайнова крива с произволен брой контролни точки
  - обикновено се използват 4

- Нека са зададени n+1 контролни точки

$$p_k = (x_k, y_k, z_k), \qquad k = 0 \div n$$

- По координатите на контролните точки се определя позиционен вектор $P(u)$
  - описва с Безие полиномна функция пътя между p_0 и p_n

$$P(u) = \sum_{k=0}^{n} p_k BEZ_{k,n}(u), \qquad 0 \le u \le 1$$

Сплайн криви на Безие

- Безие функциите $BEZ_{k,n}(u)$ са полиноми на Бернщайн

$$BEZ_{k,n}(u) = C(n,k)u^k(1-u)^{n-k}$$

където параметрите C(n,k) са биномни коефициенти: $C(n,k) = \frac{n!}{k!(n-k)!}$

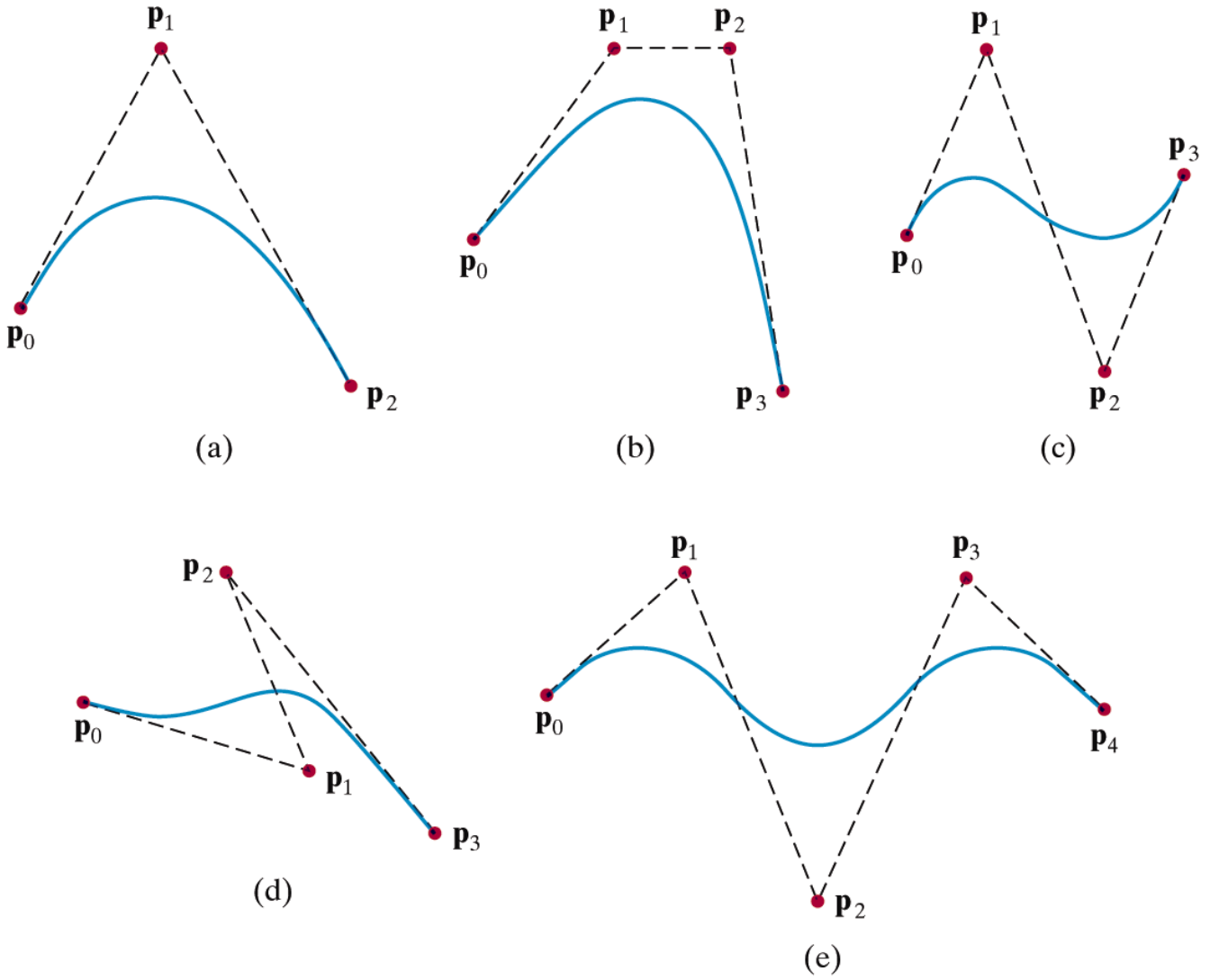
- Координати на точките от кривата

$$x(u) = \sum_{k=0}^{n} x_k BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k BEZ_{k,n}(u)$$

Сплайн криви на Безие



Сплайн криви на Безие

- Първата и последната контролни точки са първата и последната точки от кривата
  - $P(0) = p_0$
  - $P(1) = p_n$

- Кривата лежи в изпъкналата обвивка
  - тъй като всички Безие функции са положителни и сумата им е 1

$$\sum_{k=0}^{n} BEZ_{k,n}(u) = 1$$

Сплайн криви на Безие

- ***Линейна*** $\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , \ t \in [0,1]$
  - дадени са две точки
  - Безие кривата е правата между тях
    - еквивалентна на линейна интерполация
- ***Квадратична*** $\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \ , \ t \in [0,1]$
  - дадени са три точки
  - използват се в True Type шрифтове
- ***Кубична***

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , \ t \in [0,1]$$

Сплайн криви на Безие

- При 4 контролни точки
  - n = 3
  - Безие функции

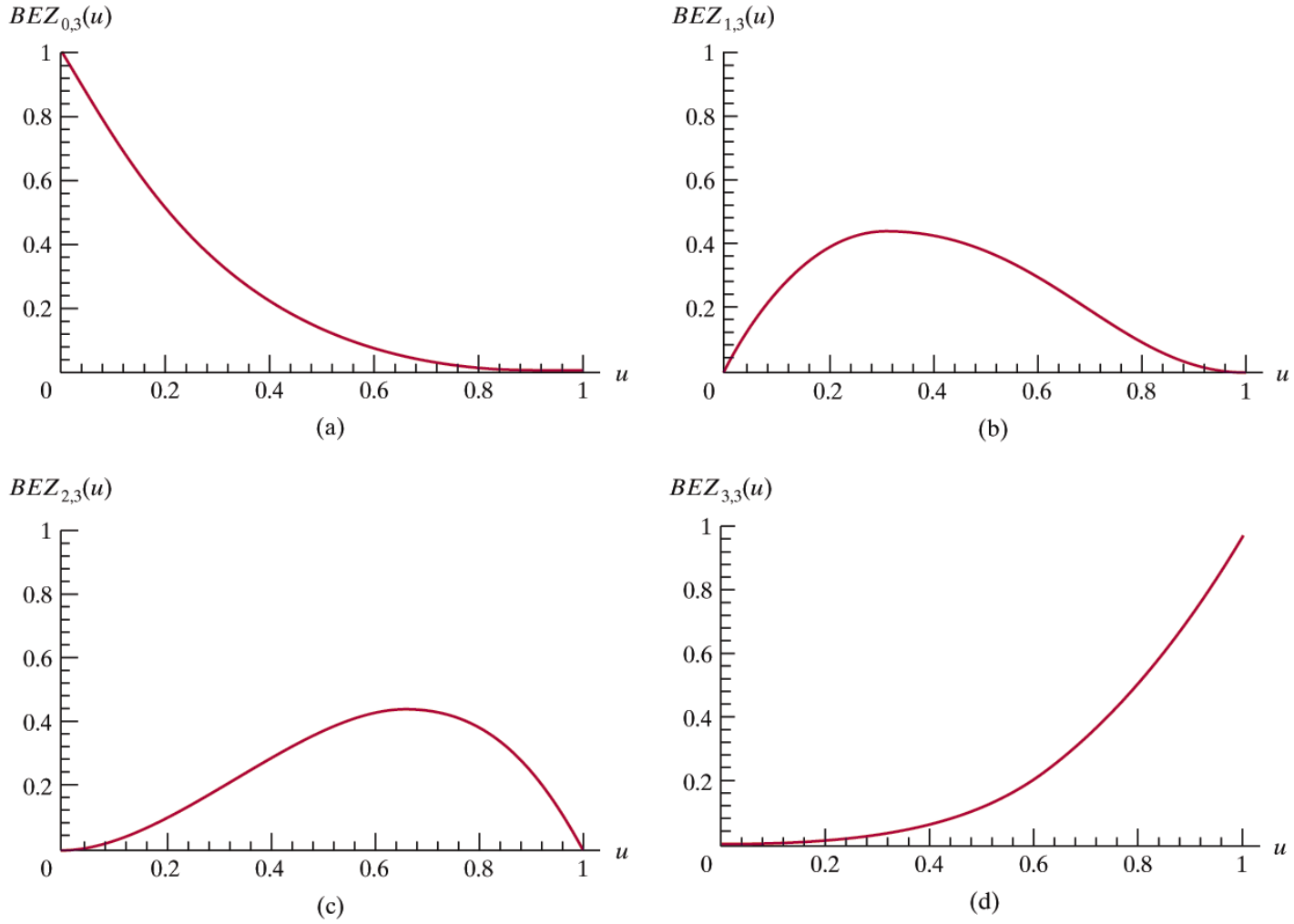
$$BEZ_{0,3} = (1-u)^3$$

$$BEZ_{1,3} = 3u(1-u)^2$$

$$BEZ_{2,3} = 3u^2(1-u)$$

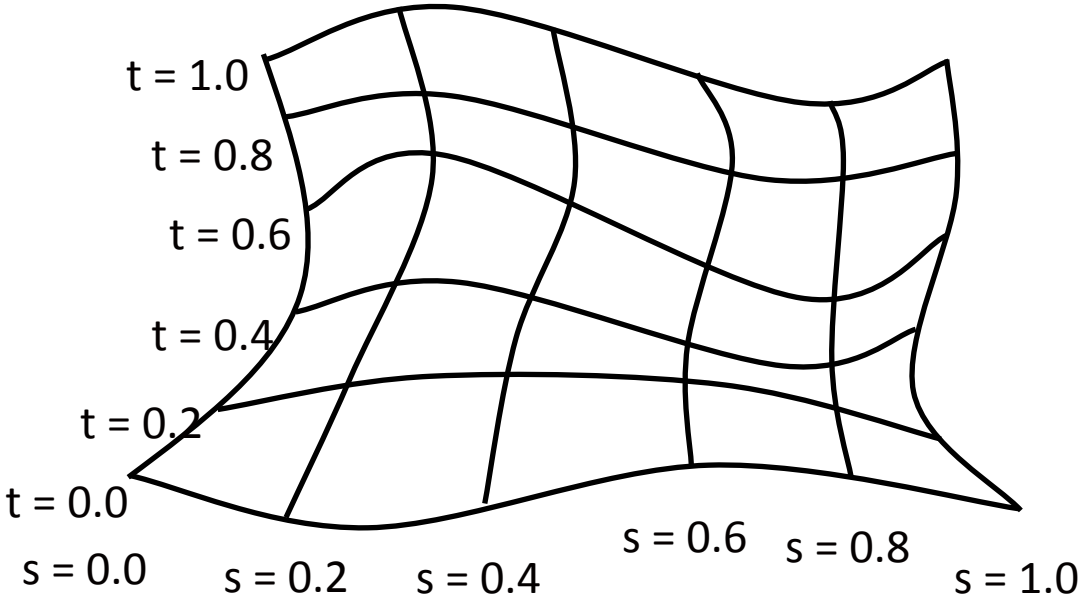
$$BEZ_{3,3} = u^3$$

Сплайн криви на Безие



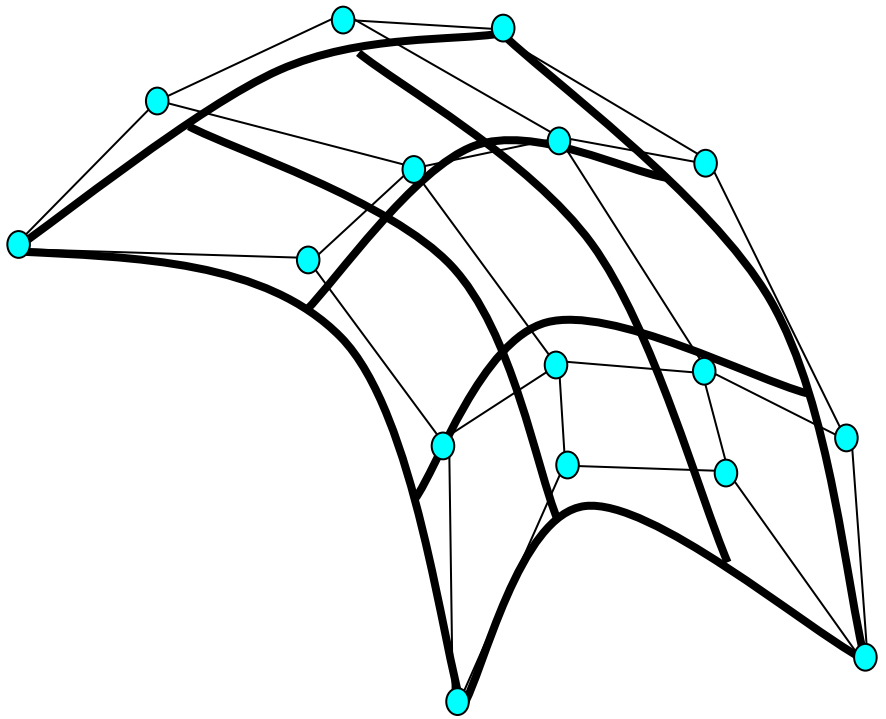
Параметрични повърхности

- ***Параметрични бикубични повърхности***
  - разширение на кривите с добавяне на още едно измерение



Безие повърхности

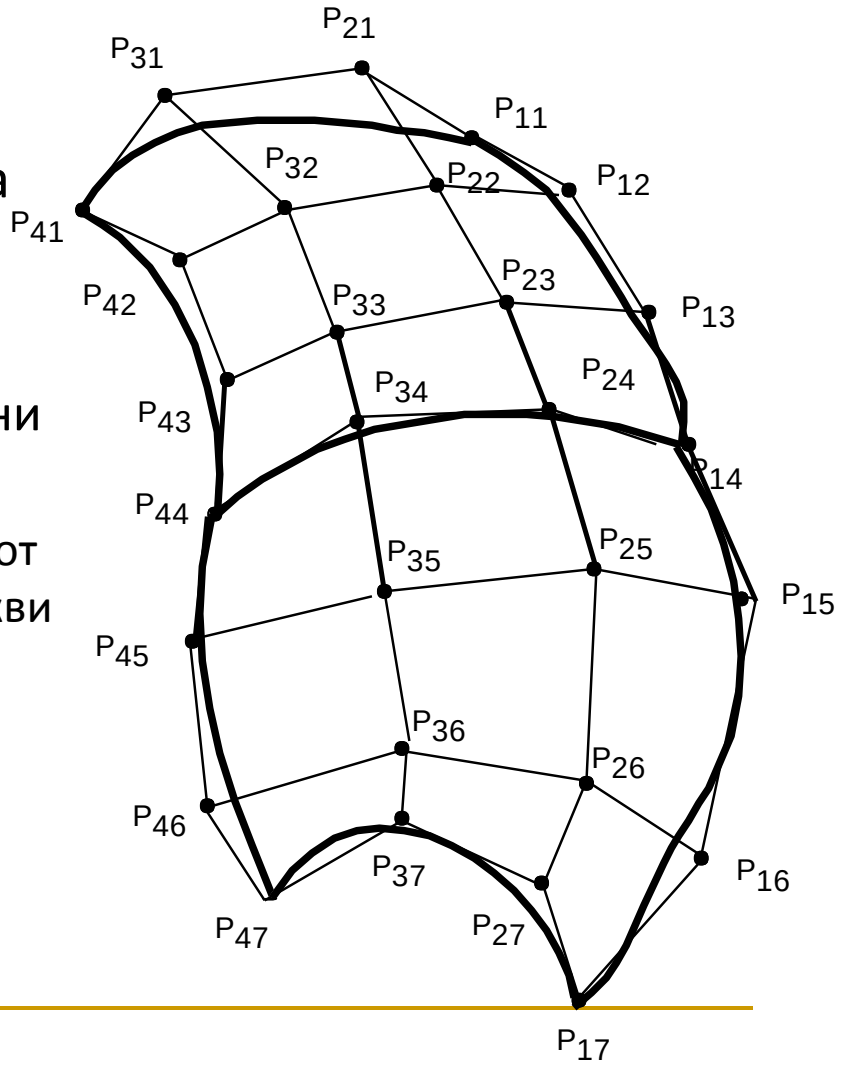
- За всяка област се изискват по 16 контролни точки
  - по 4 контролни точки в s направление
  - по 4 контролни точки в t направление



Безие повърхности

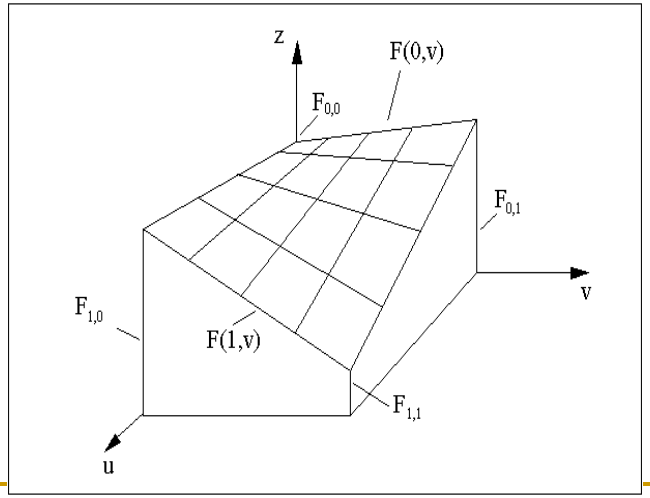
- Свързване на Безие повърхности
  - четири общи контролни точки за двете области
  - допълнително условие
    - колинеарни точки от двете страни на свързването
      - т.е. двойките линейни сегменти от колинеарни точки да имат еднакви отношения
      - за примера

$$\frac{P_{13}P_{14}}{P_{14}P_{15}} = \frac{P_{23}P_{24}}{P_{24}P_{25}} = \frac{P_{33}P_{34}}{P_{34}P_{35}} = \frac{P_{43}P_{44}}{P_{44}P_{45}}$$



Безие повърхности

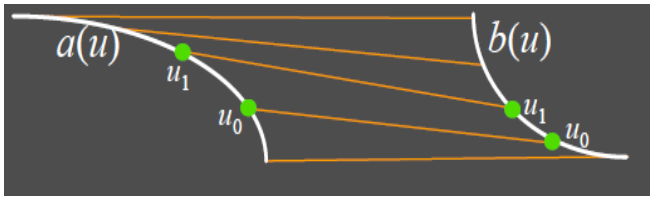
- Повърхностите се визуализират подобно на кривите
  1. Итеративно се определя уравнението на повърхността за s и t в определен интервал със зададена стъпка и изобразяване на полигони за тези области
  2. Разделяне на повърхността докато областите станат достатъчно малки



$$P(u,v) = (1-u)(1-v)P_{00} + (1-u)vP_{01} + u(1-v)P_{10} + uvP_{11}$$

Безие повърхности

- ***Ruled surface***
  - по дадени две криви се дефинира повърхност между тях

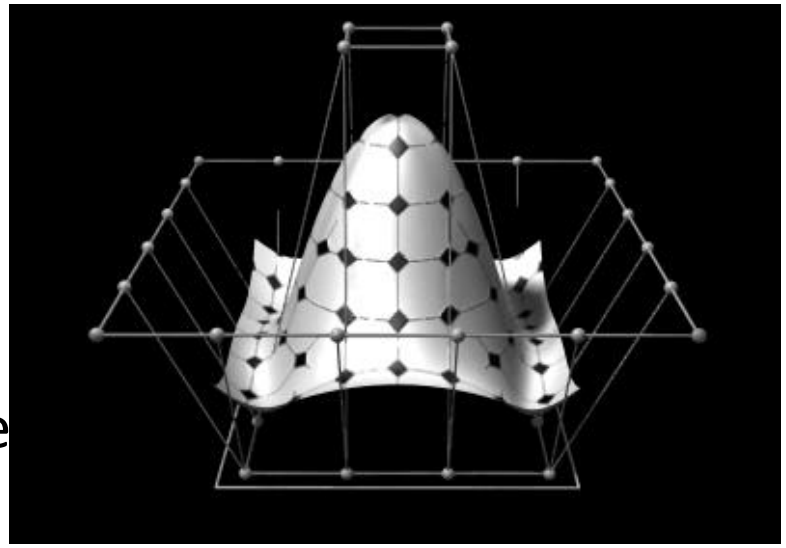


Безие повърхности

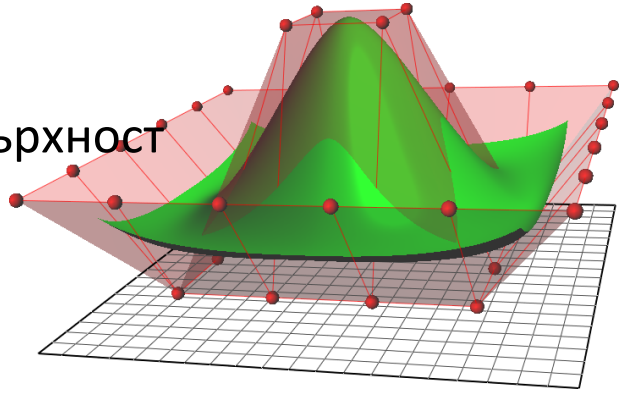
- ***Недостатъци***
  - броя контролни точки определят степените на свобода
    - промяна в една контролна точка променя цялата крива
  - при голям брой контролни точки кривата се отклонява от точки
  - не могат да се представят конични повърхности

- преодоляване на недостатъците
  - rational curves – ***B-splines***

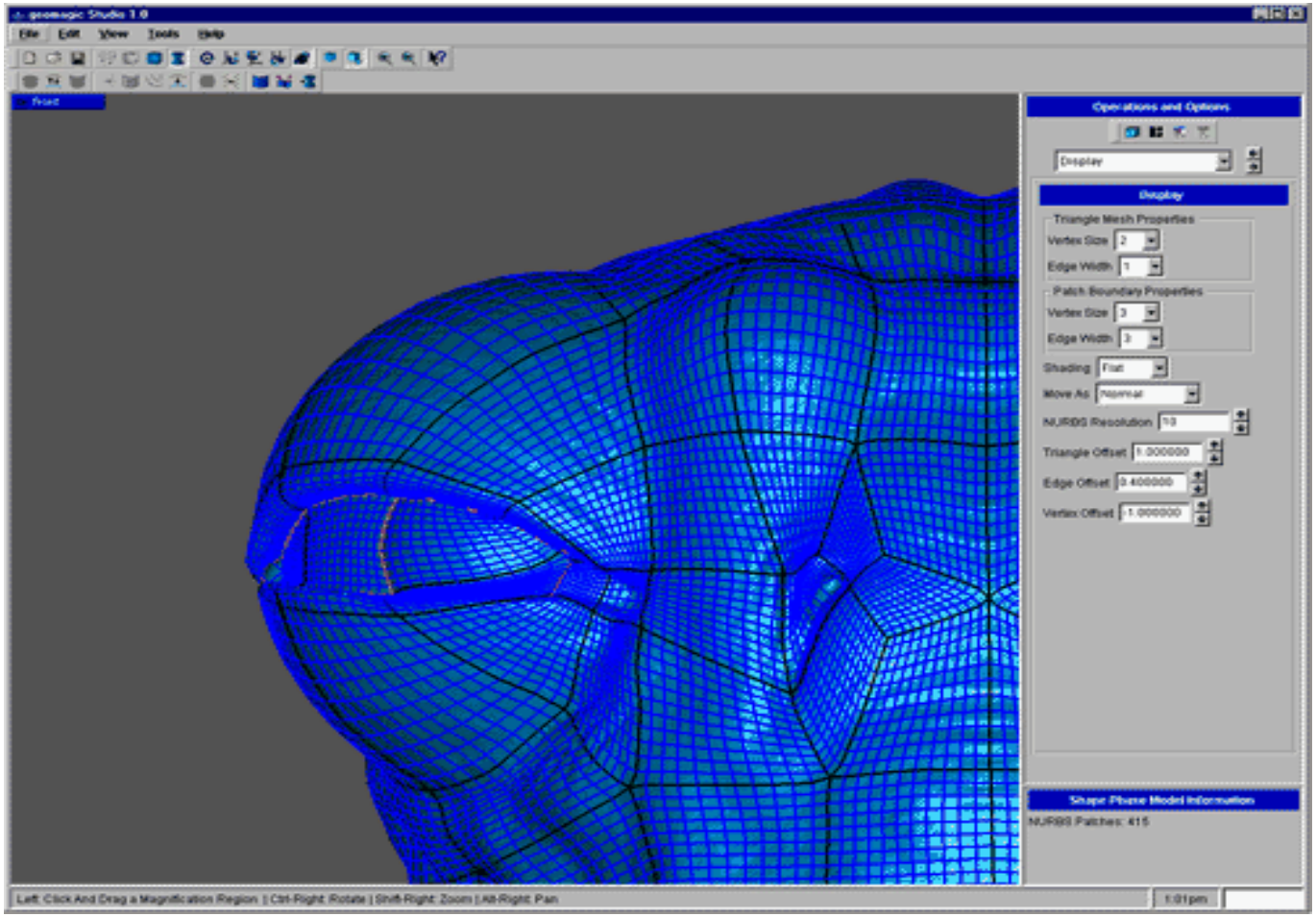
NURBS повърхности



- ***NURBS***
  - Non-Uniform Rational Basis Spline
    - Non Uniform
      - произволно желано разстояние между възлите
    - Rational
      - сплайновите функции са отношение на два полинома
    - B-Spline
      - повърхността е Безие сплайнова повърхност
  - гъвкави, мощни, сложни



NURBS примери



http://www.geomagic.com

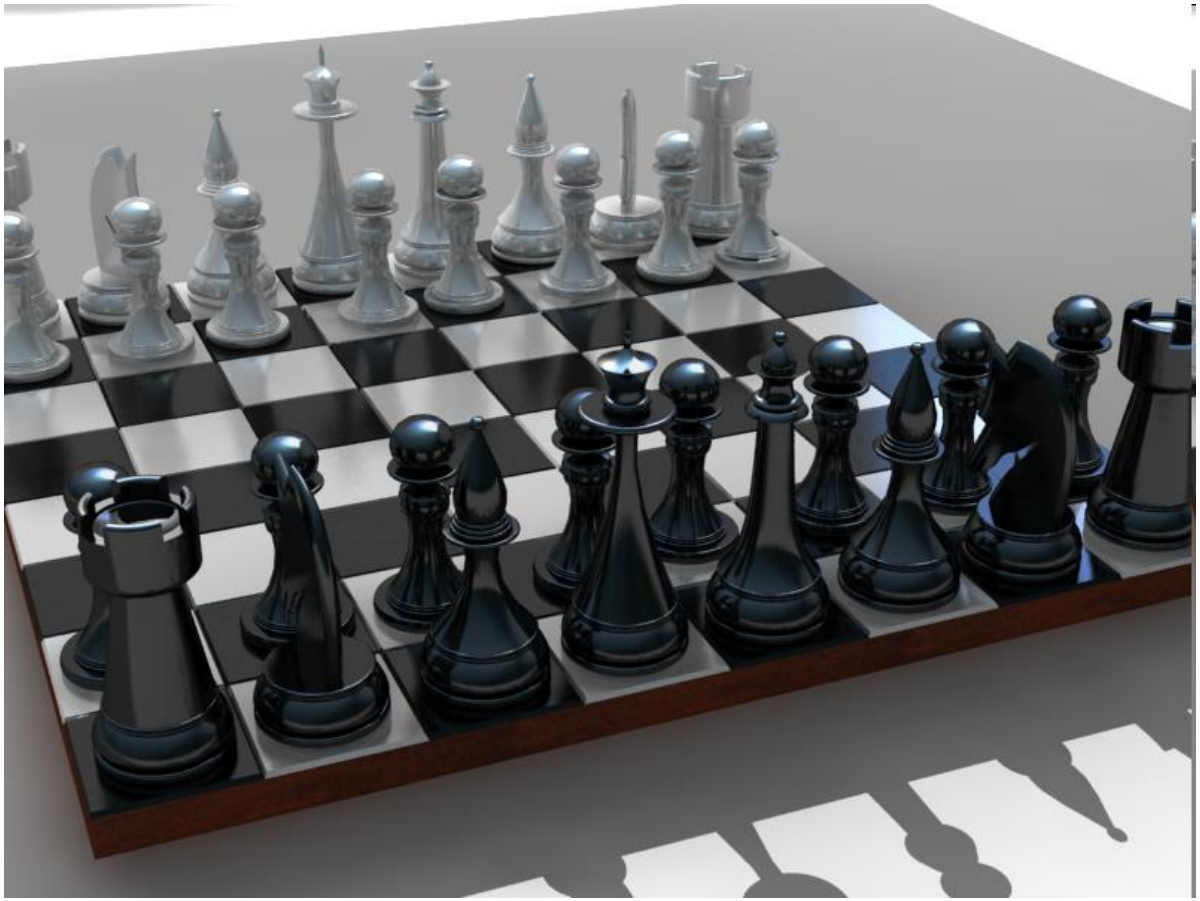
NURBS примери

http://gallery.mcneel.com

NURBS примери

http://gallery.mcneel.com

NURBS примери

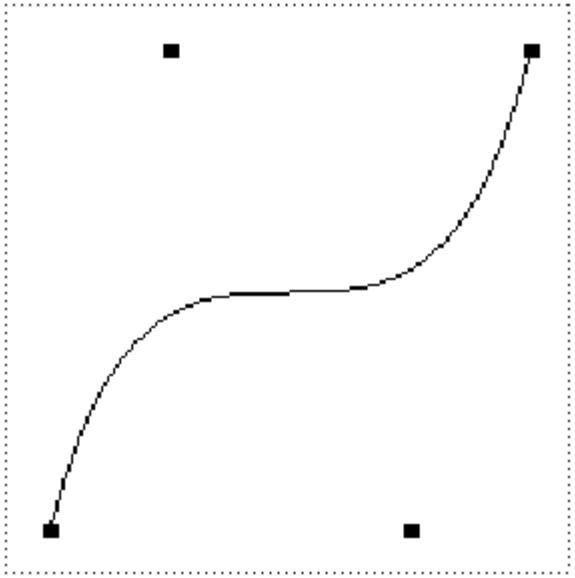


http://gallery.mcneel.com

Сплайн криви в OpenGL

- ***One-Dimensional Evaluators***
  - пример: визуализиране на Безие крива с 4 контролни точки

```
GLfloat ctrlpoints[4][3] = {{ -4.0, -4.0, 0.0},
                            { -2.0, 4.0, 0.0},
                            {2.0, -4.0, 0.0},
                            {4.0, 4.0, 0.0} };
```



Сплайн криви в OpenGL

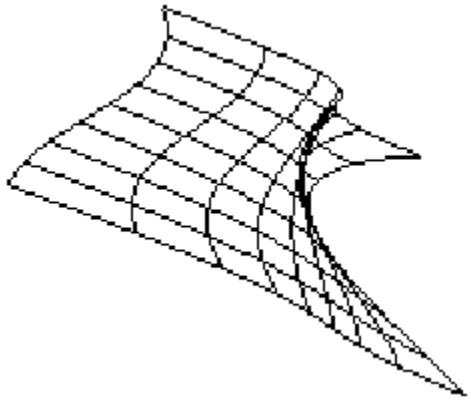
```
void myinit(void) {
glClearColor(0.0, 0.0, 0.0, 1.0);
glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4,
                                    &ctrlpoints[0][0]);
glEnable(GL_MAP1_VERTEX_3);
glShadeModel(GL_FLAT);
}
```

- GL_MAP1_VERTEX_3
  - генерира тримерни възли
    - 0: ниска стойност на параметъра u
    - 1: висока стойност на параметъра u
    - 3: брой реални числа за определяне на стойности между две контролни точки
    - 4: ред на сплайн кривата (степента+1)
    - &ctrlpoints[0][0]: указател към първата контролна точка

Сплайн криви в OpenGL

```
void display(void) {
   int i;
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glBegin(GL_LINE_STRIP);
     for (i = 0; i <= 30; i++)
       glEvalCoord1f((GLfloat) i/30.0);
   glEnd();
/* The following code displays the control points as dots. */
   glPointSize(5.0);
   glColor3f(1.0, 1.0, 0.0);
   glBegin(GL_POINTS);
     for (i = 0; i < 4; i++)
       glVertex3fv(&ctrlpoints[i][0]);
   glEnd();
   glFlush();
 }
```

Сплайн повърхнини в OpenGL



- ***Two-Dimensional Evaluators***
  - пример: визуализиране на Безие повърхност

```
GLfloat ctrlpoints[4][4][3] = {
 {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},
  {0.5, -1.5, -1.0}, {1.5, -1.5,2.0}},
 {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},
  {0.5, -0.5, 0.0},  {1.5, -0.5,-1.0}},
 {{-1.5, 0.5, 4.0},  {-0.5, 0.5, 0.0},
  {0.5, 0.5, 3.0},   {1.5, 0.5, 4.0}},
 {{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0},
  {0.5, 1.5, 0.0},   {1.5, 1.5, -1.0}} };
```

Сплайн повърхнини в OpenGL

```
void myinit(void){
 glClearColor (0.0, 0.0, 0.0, 1.0);
 glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4,
                                  &ctrlpoints[0][0][0]);
 glEnable(GL_MAP2_VERTEX_3);
 glEnable(GL_DEPTH_TEST);
 glShadeModel(GL_FLAT);
}
```

Сплайн повърхнини в OpenGL

```
void display(void) {
  int i, j;
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glPushMatrix ();
  glRotatef(85.0, 1.0, 1.0, 1.0);
  for (j = 0; j <= 8; j++) {
    glBegin(GL_LINE_STRIP);
     for (i = 0; i <= 30; i++)
      glEvalCoord2f((GLfloat)i/30.0, (GLfloat)j/8.0);
    glEnd();
    glBegin(GL_LINE_STRIP);
     for (i = 0; i <= 30; i++)
      glEvalCoord2f((GLfloat)j/8.0, (GLfloat)i/30.0);
    glEnd();
  }
 glPopMatrix(); glFlush();
}
```

Сплайн повърхнини в OpenGL

- ***Two-Dimensional Evaluators***
  - пример: визуализиране на осветена рендирана Безие крива чрез мрежа

```
void initlights(void) {
 GLfloat ambient[] = { 0.2, 0.2, 0.2, 1.0 };
 GLfloat position[] = { 0.0, 0.0, 2.0, 1.0 };
 GLfloat mat_diffuse[] = {0.6, 0.6, 0.6, 1.0 };
 GLfloat mat_specular[] = {1.0,1.0, 1.0, 1.0 };
 GLfloat mat_shininess[] = { 50.0 };
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glLightfv(GL_LIGHT0,GL_AMBIENT, ambient);
 glLightfv(GL_LIGHT0,GL_POSITION, position);
 glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,mat_diffuse);
 glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,mat_specular);
 glMaterialfv(GL_FRONT_AND_BACK,GL_SHININESS,mat_shininess);
}
```

Сплайн повърхнини в OpenGL

```
void display(void) {
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glPushMatrix();
 glRotatef(85.0, 1.0, 1.0, 1.0);
 glEvalMesh2(GL_FILL, 0, 8, 0, 8);
 glPopMatrix();
 glFlush();
}

 void myinit(void) {
 glClearColor (0.0, 0.0, 0.0, 1.0);
 glEnable(GL_DEPTH_TEST);
 glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,0, 1, 12, 4,
                                    &ctrlpoints[0][0][0]);
 glEnable(GL_MAP2_VERTEX_3);
 glEnable(GL_AUTO_NORMAL);
 glMapGrid2f(8, 0.0, 1.0, 8, 0.0, 1.0);
 initlights();
}
```

NURBS в OpenGL

```
gluNurbsSurface (GLUnurbs *nurb,
                 Glint sKnotCount,
                 GLfloat *sKnots,
                 Glint tKnotCount,
                 GLfloat *tKnots,
                 Glint sStride,
                 Glint tStride,
                 GLfloat *control,
                 Glint sOrder,
                 Glint tOrder,
                 GLenum type)
```

NURBS в OpenGL

```
gluNurbsSurface(GLUnurbs *nurb,
                GLint sKnotCount,
                GLfloat *sKnots,
                GLint tKnotCount,
                GLfloat *tKnots,
                GLint sStride,
                GLint tStride,
                GLfloat *control,
                GLint sOrder,
                GLint tOrder,
                GLenum type)
```

- *nurb*
  - специфицира NURBS обект
  - създаден с gluNewNurbsRenderer
- *sKnotCount*
  - брой възли в направление *s*
- *sKnots*
  - масив от *s* възли
- *tKnotCount*
  - брой възли в направление *t*
- *tKnots*
  - масив от *t* възли
- *sStride*
  - отместване между контролните точки в направление *s*
- *tStride*
  - отместване между контролните точки в направление *t*
- *control*
  - масив с контролни точки
- sOrder
  - степен на NURBS по s
- tOrder
  - степен на NURBS по t
- type
  - вид на повърхнината

NURBS в OpenGL

- GLUT предоставя интерфейс за NURBS създаден върху функцията за evaluator
  - пример: визуализиране на NURBS повърхност

```
include <GL/glu.h>
#include <stdlib.h>
#include <stdio.h>
GLfloat ctlpoints[4][4][3];
GLUnurbsObj *theNurb;
```

NURBS в OpenGL

```
void init_surface(void) {
 int u, v;
 for (u = 0; u < 4; u++) {
   for (v = 0; v < 4; v++) {
     ctlpoints[u][v][0] = 2.0*((GLfloat)u - 1.5);
     ctlpoints[u][v][1] = 2.0*((GLfloat)v - 1.5);
     if ( (u == 1 || u == 2) && (v == 1 || v == 2))
        ctlpoints[u][v][2] = 3.0;
     else
        ctlpoints[u][v][2] = -3.0;
   }
  }
}
```

NURBS в OpenGL

```
void myinit(void) {
   GLfloat mat_diffuse[]  = { 0.7, 0.7, 0.7, 1.0 };
   GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat mat_shininess[]= { 100.0 };
   glClearColor (0.0, 0.0, 0.0, 1.0);

   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

   glEnable(GL_LIGHTING);     glEnable(GL_LIGHT0);
   glDepthFunc(GL_LEQUAL);    glEnable(GL_DEPTH_TEST);
   glEnable(GL_AUTO_NORMAL); glEnable(GL_NORMALIZE);

   init_surface();
   theNurb = gluNewNurbsRenderer();
```

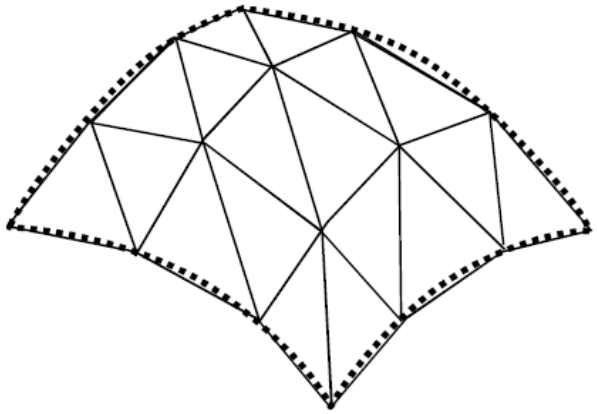
NURBS в OpenGL

```
gluNurbsProperty(theNurb, GLU_SAMPLING_TOLERANCE, 25.0);
gluNurbsProperty(theNurb, GLU_DISPLAY_MODE, GLU_FILL);
}

void display(void) {
  GLfloat knots[8] = {0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0};
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glRotatef(330.0, 1.,0.,0.); glScalef (0.5, 0.5, 0.5);
  gluBeginSurface(theNurb);
    gluNurbsSurface(theNurb, 8, knots, 8, knots, 4 * 3,
        3, &ctlpoints[0][0][0], 4, 4, GL_MAP2_VERTEX_3);
  gluEndSurface(theNurb);
  glPopMatrix();
  glFlush();
}
```

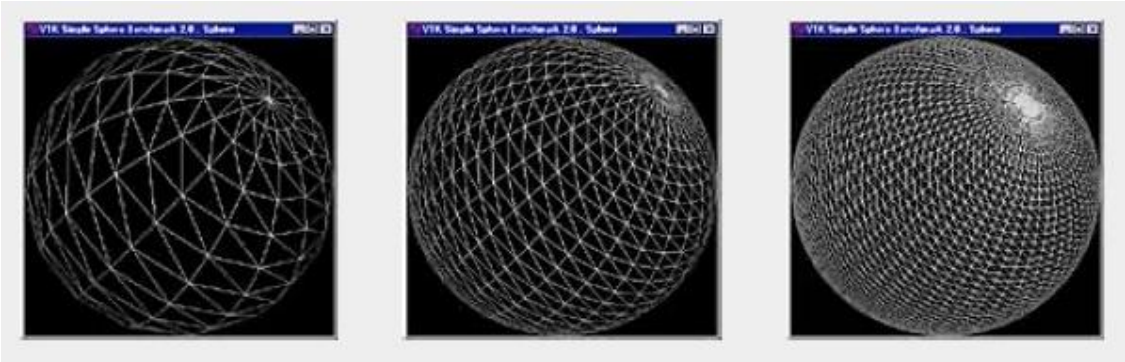
Параметрични повърхности

- ***Tessellation***
  - бикубичните повърхности не могат директно да се манипулират при синтеза на изображения
  - повърхността се разделя на триъгълници



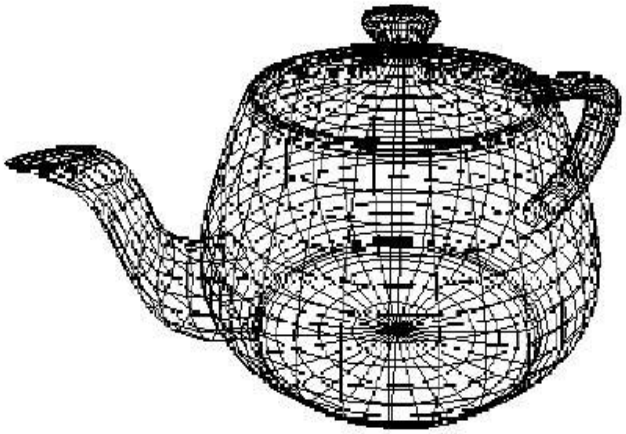
Параметрични повърхности

- ***Tessellation***
  - Подходи
    - равномерно дискретизиране на параметричното пространство
    - адаптивно разделяне на базата на кривината на повърхността



Параметрични повърхности

- ***Tessellation***
  - пример: Utah Teapot
    - моделът е създаден през 1975 от Martin Newell от Университета в Юта

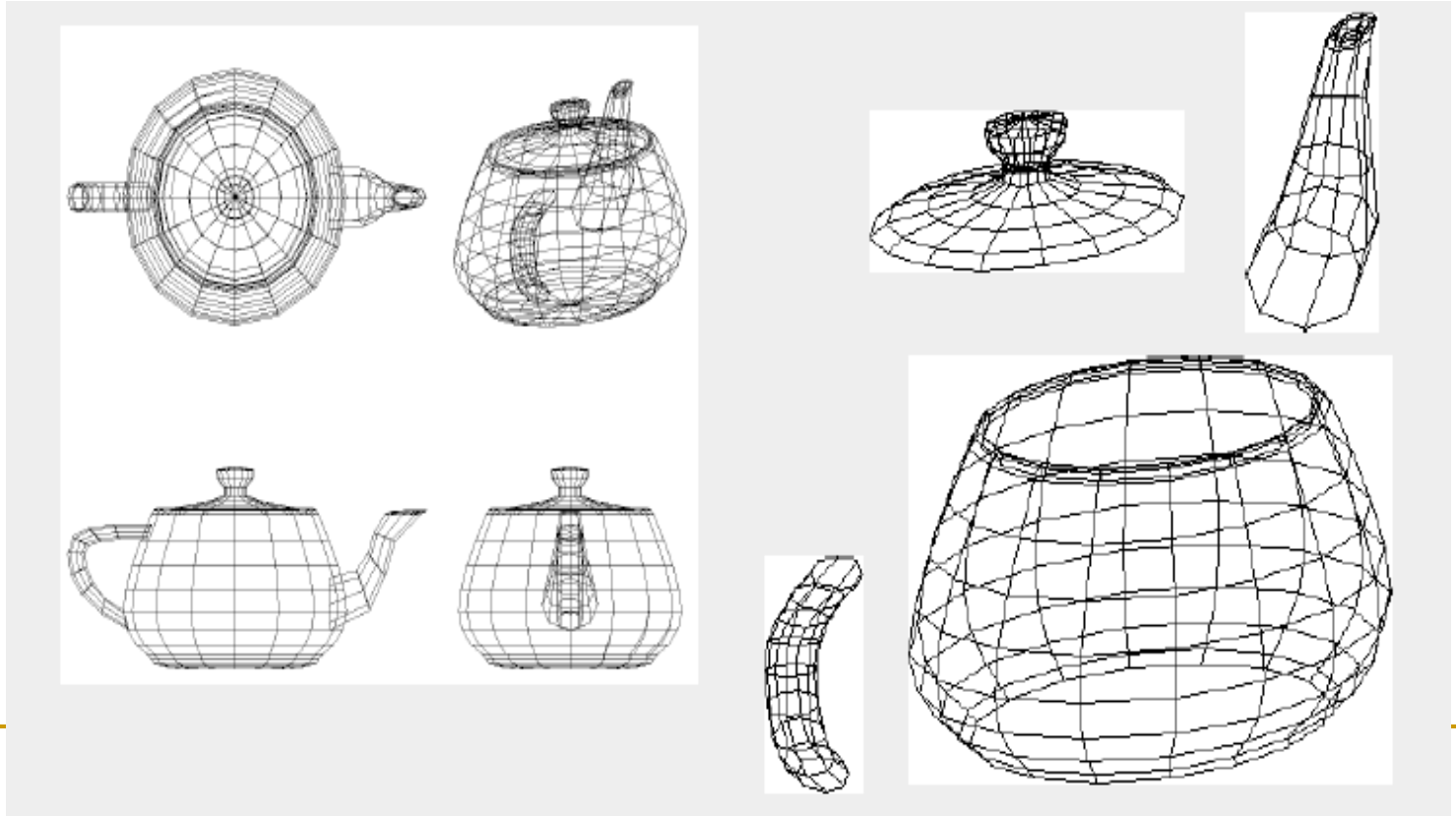


    - 32 Bezier surface patches
      - 10 уникални
      - останалите огледални образи

- използва се като геометричен примитив в OpenGL

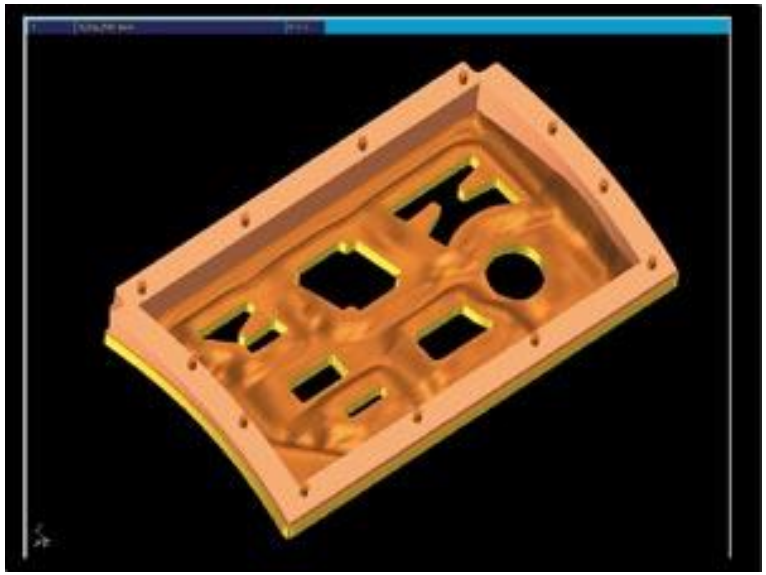
Параметрични повърхности

- ***Tessellation***
  - пример: Utah Teapot

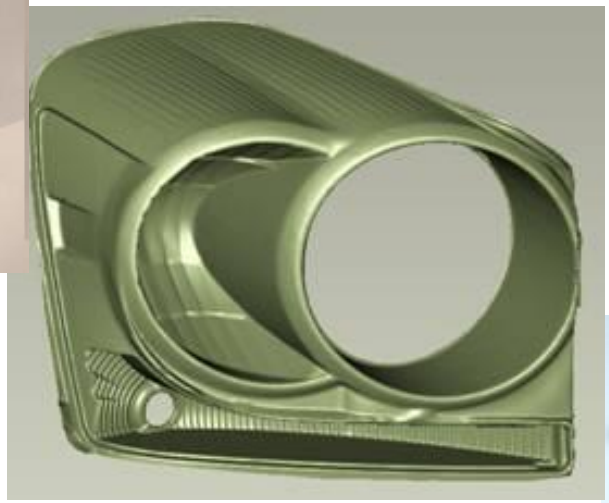


Изрязващи криви

- Задават отвори в повърхността
  - изрязващата крива е NURBS крива върху NURBS повърхността
  - повърхността се визуализира навсякъде, освен във вътрешността на изрязващите криви



Изрязващи криви



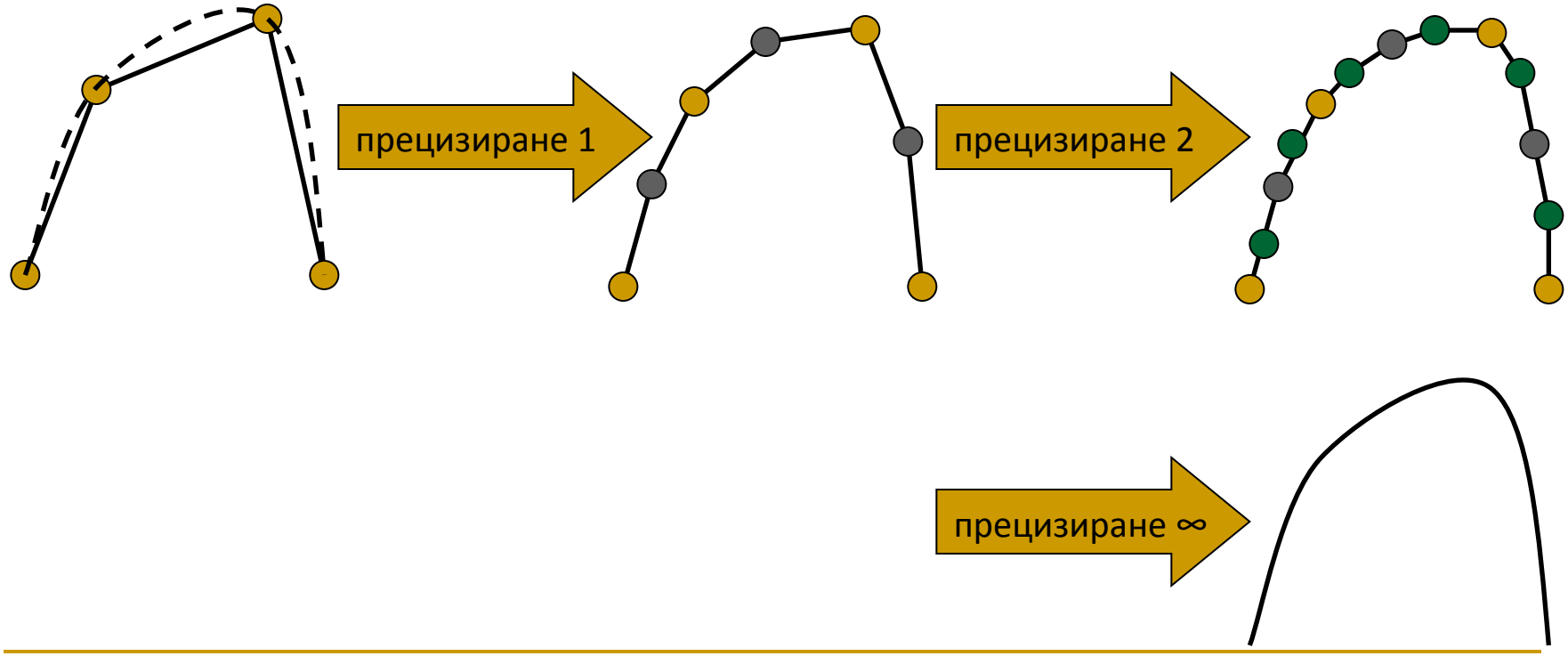
Разделящи повърхности

- Проблем при повърхностите
  - промяна на разделителната способност за част от повърхността
    - за да се визуализират повече детайли само в част от дадена област трябва да се опише и визуализира цялата област

- ***Разделящи повърхности***
  - позволяват локален контрол на мрежите
    - по-голяма гъвкавост при моделирането на обектите

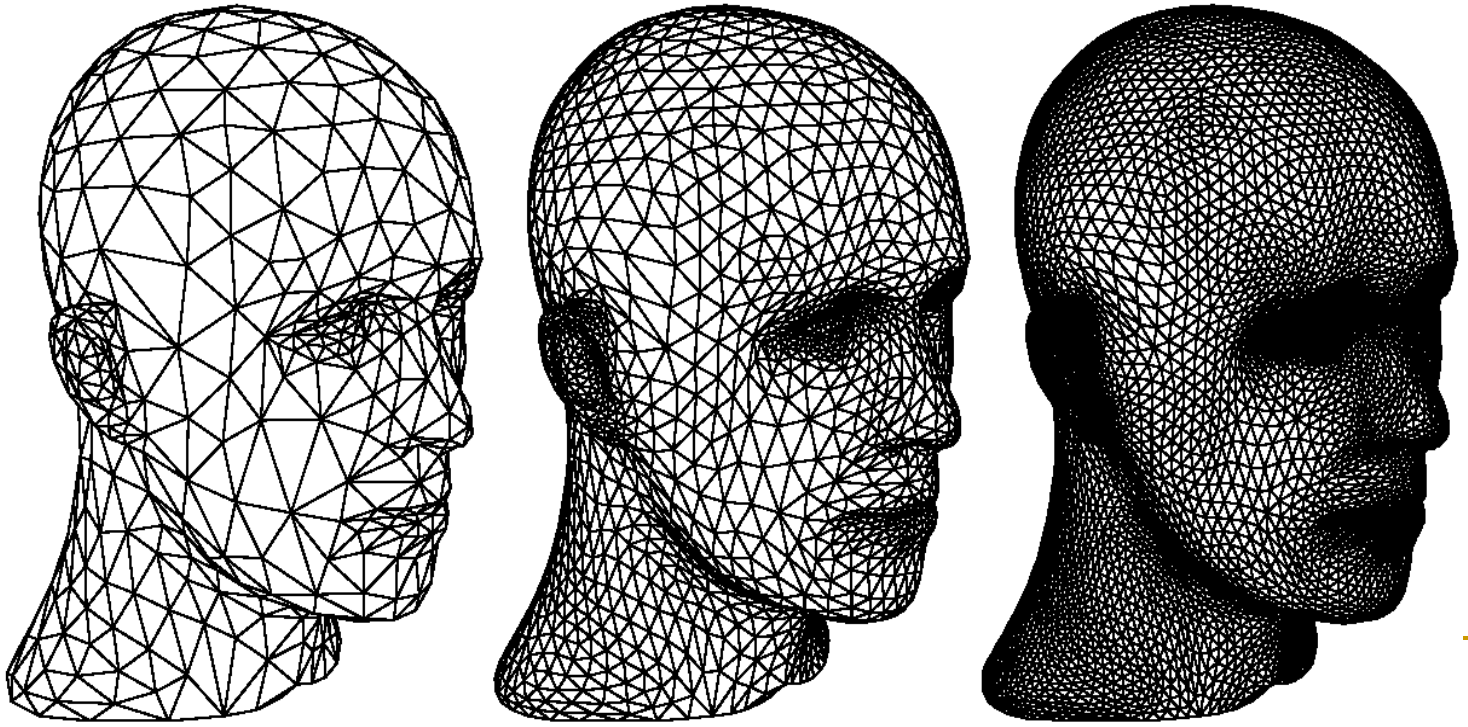
Разделящи повърхности

- Рекурсивно разделяне на областта от повърхността за визуализиране на по-фина резолюция



Стандартно разделяне

- При зададени начални контролни точки се прилага рекурсивно разделяне докато се достигне желана гладкост

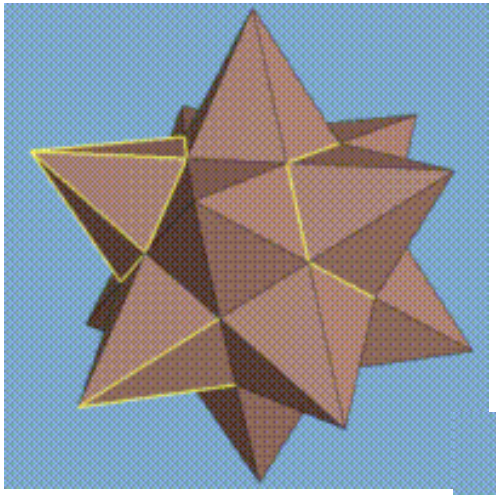


Адаптивно разделяне

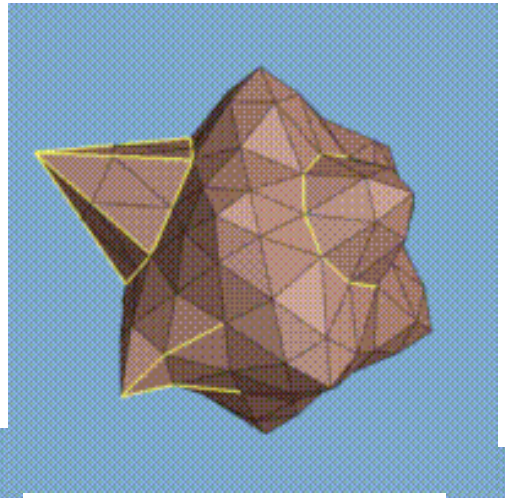
- Обикновено някои региони в повърхността имат по-голяма кривина и следва да бъдат разделени допълнително на повече области от останалите региони

- ***Адаптивно разделяне***
  - региони с голяма кривина
  - региони, в които се цели по-фина разделителна способност

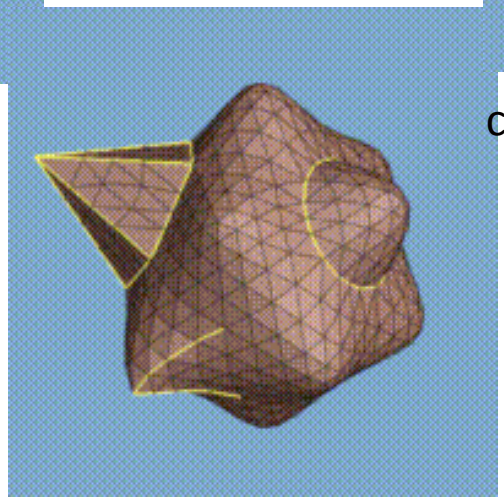
Адаптивно разделяне



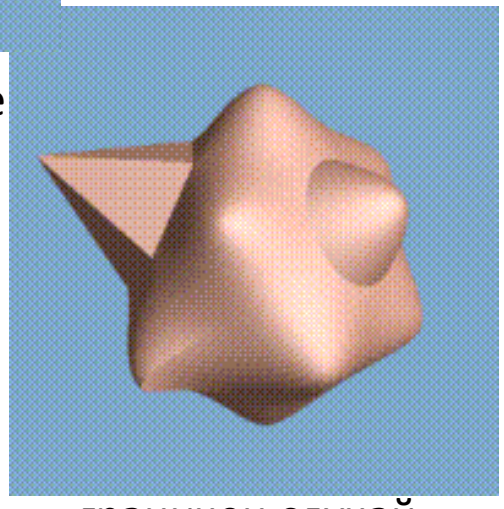
оригинална мрежа



след едно разделяне



след две разделяния



граничен случай
(безкраен брой разделяния)

http://grail.cs.washington.edu/projects/subdivision/

Адаптивно разделяне

Geri’s Game (1997), Pixar Animation Studios

КРАЙ

Следваща тема:

Осветеност