THREE.JS – IMPORT OBJEKTOV A VIAZANIE POHYBU OBJEKTU PO KRIVKE

doc. Ing. Branislav Sobota, PhD.
Ing. Marián Hudák, Ing. Miriama Mattová, Ing. Lenka Bubeňková
Katedra počítačov a informatiky, FEI TU v Košiciach

C 09

© 2024

Fakulta elektrotechniky a informatiky

Počítačová Grafika

Ciele cvičenia

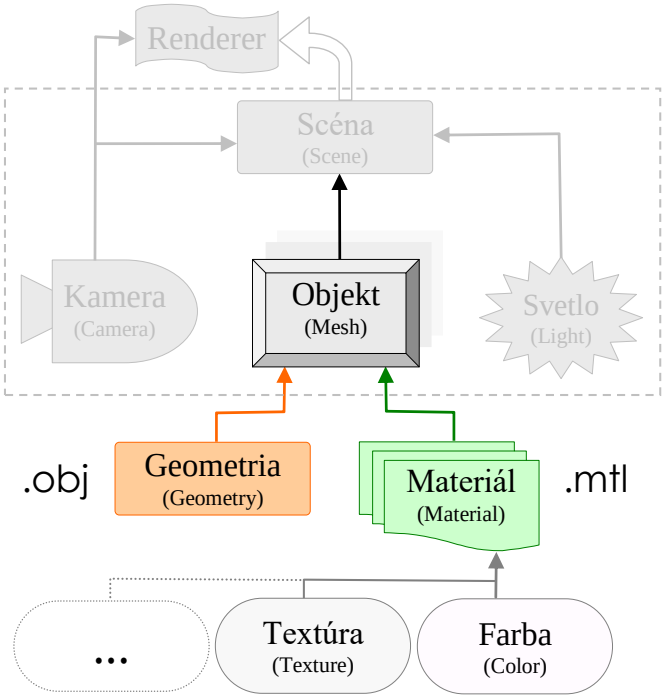
- Three.js - importovanie objektov
- Three.js - metódy pre importovanie OBJ modelov
- Three.js - implementácia krivky v 3D.
- Three.js - viazanie pohybu objektu po krivke.

1. *THREE.JS* – PRÍPRAVA BALÍČKA

- **Úloha:** Stiahnite si balíček „**Threejs_Import_a_pohyb_po_krivke.zip**“ z portálu Moodle KPI a predmetu Počítačová grafika.
- Obsah balíčka skopírujte do vášho projektu aby štruktúra vyzerala nasledovne:
  - `WebGL_getStart`
    - `> css`
    - `> js`
    - `>> threejs`
    - `>> ThreeImport.js`
    - `> models`
    - `>> car`
    - `>>> Pony_cartoon.obj`
    - `>>> Pony_cartoon.mtl`
    - `>>> Body_dDo_d_orange.jpg`
    - `>>> . . .`
    - `>> lirkis_car`
    - `>>> lirkis_car.obj`
    - `>>> lirkis_car.mtl`
    - `> texture`
    - `index.html`

1. THREE.JS – IMPORTOVANIE OBJEKTOV

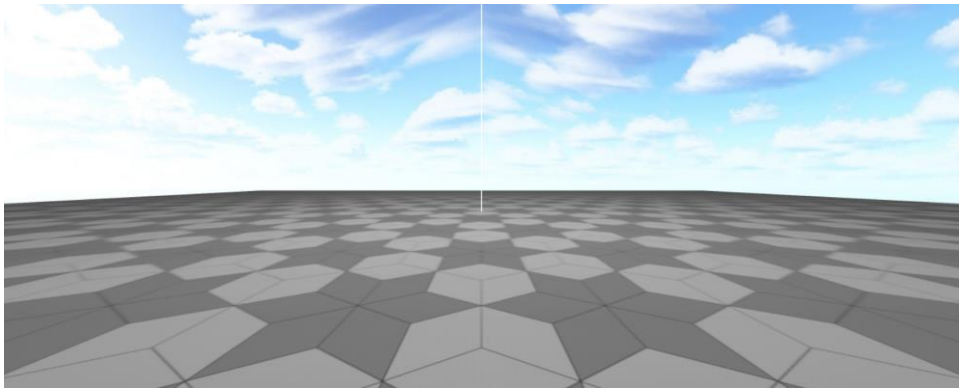
- Najčastejšie používaným formátom pre *Three.js* pre prenos geometrie objektu je formát .***OBJ*** a pre prenos materiálov je formát .***MTL***. Filozofia je daná už skôr prezentovanou štruktúrou.



1. *THREE.JS* – PRÍPRAVA SCÉNY PRE IMPORT OBJEKTOV

- **Úloha :** Vytvorte primárnu scénu pre precvičenie importu objektov.

1. Otvorte si skript „**ThreeImport.js**“ a html súbor „**index.html**“ vo vašom vývojovom prostredí.
2. V HTML súbore „**index.html**“ pribudli dve cesty pre skripty spracovávajúce formáty ***OBJ*** a ***MTL***.
   Tieto formáty reprezentujú : ***OBJ*** - formát geometrie a ***MTL*** - formát materiálu
3. Spustite si vizualizáciu počiatočnej scény vo webovom prehliadači.



2. THREE.JS – METÓDY PRE IMPORTOVANIE OBJ MODELOV

- Importovať modely v tomto formáte je možné viacerými spôsobmi v závislosti od požadovanej kvality a samozrejme následnej zložitosti importovaného modelu.
- Podľa typu použitého osvetľovacieho modelu (pozri predchádzajúce cvičenie) si ukážeme ako je možné importovať objekt :
  - ***štandardne*** (základne, basic)
  - a s použitím ***Phongovho osvetľovacieho modelu***.

2. THREE.JS – ŠTANDARDNÁ METÓDA PRE IMPORTOVANIE OBJ MODELU

- Pri tomto type importu sa využíva len **základný** (basic, standard) **osvetľovací model**.
- Základom je funkcia **loadOBJectsStandard()**.
- Vstupné parametre metódy **loadOBJectsStandard()**:

  - `loadOBJectsStandard( x,y,z, path, scalex, scaley, scalez, texturePath, colorMaterial)`

    - **x,y,z** – pozícia 3D OBJ objektu : float
    - **path** – cesta k súboru objektu : string
    - **scalex, scaley, scalez** – mierka objektu : float
    - **texturePath** – cesta k súboru textúry : string
    - **colorMaterial** – farba implicitného materiálu : float (RGB) | | string

2. THREE.JS – ŠTANDARDNÁ METÓDA PRE IMPORTOVANIE OBJ MODELU

1. V skripte „**ThreeImport.js**“ vložte nasledujúcu metódu:

```
function loadOBJectsStandard(x,y,z, path, scalex, scaley, scalez,
texturePath, colorMaterial){
 var loader = new THREE.OBJLoader();
 var textureSurface = new THREE.TextureLoader().load(texturePath);
 var material = new THREE.MeshStandardMaterial({
                    color: colorMaterial,
                    map: textureSurface,
                    roughness : 0.05,
                    metalness: 0.45
     });
 loader.load( path, function ( object ) {
   object.traverse( function ( node ) {
     object.position.set(x,y,z);
     object.material = material;
     object.scale.set(scalex,scaley,scalez);
     if ( node.isMesh ) node.material = material;
   });
 scene.add( object );
 });
}
```

2. THREE.JS – ŠTANDARDNÁ METÓDA PRE IMPORTOVANIE OBJ MODELU

2. V skripte „**ThreeImport.js**“ vložte v metóde **addObjects()** nasledujúcu implementáciu :

```
loadOBJectsStandard( 2,-0.5,0,
                     'models/car/Pony_cartoon.obj',
                     0.003,0.003,0.003,
                     "models/car/Body_dDo_d_orange.jpg",
                     "white"
                    );
```

3. Overte správnosť implementácie spustením vizualizácie vo webovom prehliadači.

2. THREE.JS – METÓDA PRE IMPORTOVANIE OBJ MODELU S PODPOROU PHONGOVHO OSVETĽOVACIEHO MODELU

- Pri tomto type importu je potrebné pripraviť a nastaviť parametre pre prácu s ***Phongovým osvetľovacím modelom***.
- Základom je funkcia **loadOBJectsPhong()**.
- Vstupné parametre metódy **loadOBJectsPhong()** sú ekvivalentné ako pri metóde standard:

  - `loadOBJectsPhong( x,y,z, path, scalex, scaley, scalez, texturePath, colorMaterial)`

    - **`x,y,z`** – pozícia 3D OBJ objektu : float
    - **`path`** – cesta k súboru objektu : string
    - **`scalex, scaley, scalez`** – mierka objektu : float
    - **`texturePath`** – cesta k súboru textúry : string
    - **`colorMaterial`** – farba implicitného materiálu : float (RGB) | | string

2. THREE.JS – METÓDA PRE IMPORTOVANIE OBJ MODELU S PODPOROU PHONGOVHO OSVETĽOVACIEHO MODELU

1. V skripte „**ThreeImport.js**“ vložte nasledujúcu metódu:

```
function loadOBJectsPhong(x,y,z, path, scalex, scaley, scalez,
texturePath, colorMaterial){
 var loader = new THREE.OBJLoader();
 var textureSurface = new THREE.TextureLoader().load(texturePath);
 var material = new THREE.MeshPhongMaterial({
                    color: colorMaterial,
                    map: textureSurface
     });
 loader.load( path, function ( object ) {
   object.traverse( function ( node ) {
     object.position.set(x,y,z);
     object.material = material;
     object.scale.set(scalex,scaley,scalez);
     if ( node.isMesh ) node.material = material;
   });
   scene.add( object );
  });
}
```

2. THREE.JS – METÓDA PRE IMPORTOVANIE OBJ MODELU S PODPOROU PHONGOVHO OSVETĽOVACIEHO MODELU

2. V skripte „ThreeImport.js“ vložte v metóde addObjects() nasledujúcu implementáciu :

```
loadOBJectsPhong( -2,-0.5,0,
                  'models/car/Pony_cartoon.obj',
                  0.003,0.003,0.003,
                  "models/car/Body_dDo_d_orange.jpg",
                  "white");
```

3. Overte správnosť implementácie spustením vizualizácie vo webovom prehliadači.

2. THREE.JS – POROVNANIE VIZUÁLNYCH ROZDIELOV

- Pozrite si vizuálne rozdiely medzi jednotlivými modelmi pri pohybe kamerou po scéne.

2. THREE.JS – ŠTANDARDNÁ METÓDA PRE IMPORTOVANIE VLASTNÉHO OBJ+MTL MODELU

- Pri tomto type importu sa využívajú dve funkcie.
- Prvou je funkcia **MTLLoader()** pre načítanie materiálu zo súboru **.MTL** (cestu definuje parameter `MTLpath`).
- Druhou je funkcia **OBJLoader()** pre načítanie geometrie modelu (cestu definuje parameter `OBJpath`).
- Po načítaní materiálu/ov sa začne načíta geometria. Objektu z **OBJLoader()** je potrebné pred načítaním samotnej geometrie priradiť už načítaný materiál.
- Po ukončení načítania je potrebné vložiť objekt do scény pomocou funkcie **scene.add(objekt)**.
- Implementácia je zobrazená na nasledujúcom slajde.

2. THREE.JS – THREE.JS – ŠTANDARDNÁ METÓDA PRE IMPORTOVANIE VLASTNÉHO OBJ+MTL MODELU

1. V skripte „**ThreeImport.js**“ vložte nasledujúcu metódu:

```
function loadObjWithMTL(objPath, MTLpath, scalex, scaley, scalez,
posX, posY, posZ){
    // ak nie je potrebná globálna premenná tu definujte objekt objcar;
    var mtlLoader = new THREE.MTLLoader();

    mtlLoader.load( MTLpath, function( materials ) {
        materials.preload();

        var objLoader = new THREE.OBJLoader();
        objLoader.setMaterials(materials);
        objLoader.load(objPath, function(object)
        {
            objcar = object;
            objcar.position.set(posX,posY,posZ);
            objcar.scale.set(scalex,scaley,scalez);
            scene.add( objcar );
        });
    });
}
```

2. THREE.JS – ŠTANDARDNÁ METÓDA PRE IMPORTOVANIE VLASTNÉHO OBJ+MTL MODELU

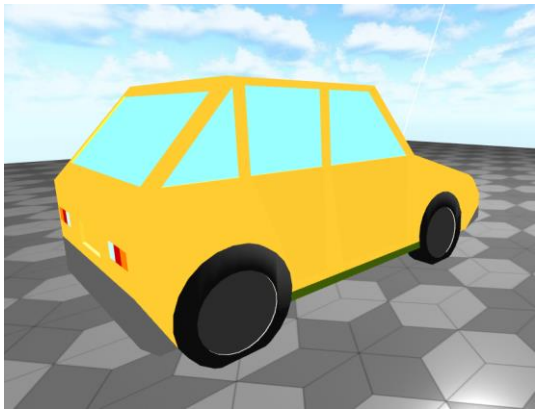
2. V skripte „**ThreeImport.js**“ vložte na začiatok premennú:

```
var objcar;
```

3. V skripte „**ThreeImport.js**“ vložte v metóde **addObjects()** nasledujúcu implementáciu :

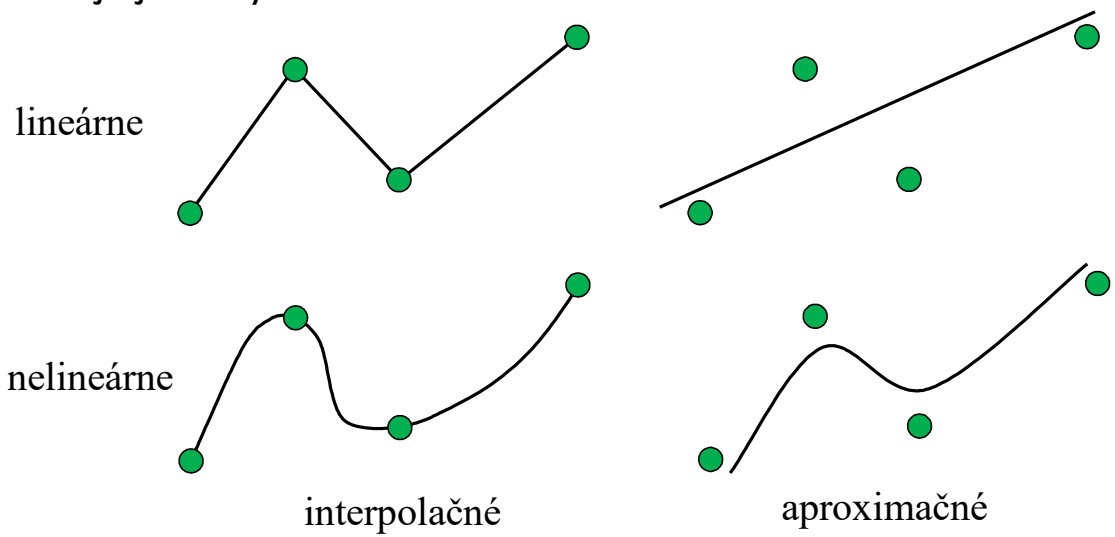
```
loadObjWithMTL("models/lirkis_car/lirkis_car.obj",
               "models/lirkis_car/lirkis_car.mtl",
               0.5,0.5,0.5,
               0,-0.2,0);
```

4. Overte správnosť implementácie spustením vizualizácie vo webovom prehliadači.



3. THREE.JS – IMPLEMENTÁCIA KRIVKY V 3D

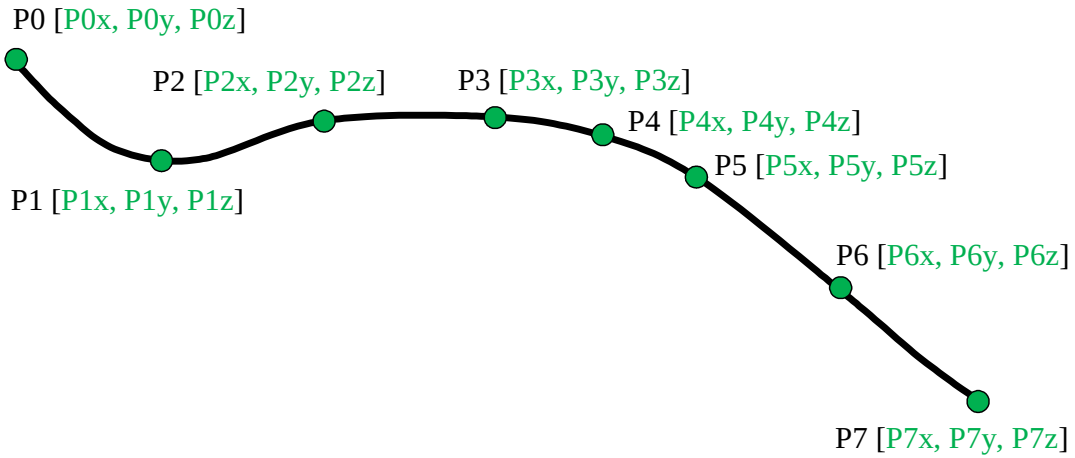
- **Každá krivka musí mať svoj matematický popis**, aby bolo možné vykonávať rôzne operácie s takouto krivkou alebo s jej časťou, ako sú napríklad posuny, rotácie, alebo zmena mierky.
- **Krivku definujeme ako 1D útvar** definovaný vrcholmi a hranami tzv. ***polyline*.**
- Podľa typov hrán (segmentov) rozoznávame krivky: ***lineárne*** a ***nelineárne***.
- Podľa vplyvu vrcholov na tvar krivky ich delíme na: ***interpolačné*** (vrcholy sú súčasťou krivky) a ***aproximačné*** (vrcholy nemusia byť súčasťou krivky, ale vplývajú na jej tvar).



3. *THREE.JS* – IMPLEMENTÁCIA *KRIVKY V 3D*

- ***Three.js* podporuje niekoľko typov kriviek**, z ktorých veľké využitie majú najmä **nelineárne *spline*-ové** krivky.
- Jednou z používaných funkcií v *Three.js* je **THREE.CatmullRomCurve3().**
- Je to 3D interporlačná spline krivka vytvorená ***Catmull-Romovým* algoritmom**. Definuje sa pomocou riadiacich bodov, ktorými prechádza.

```
var curve = new THREE.CatmullRomCurve3( [
  new THREE.Vector3( P0x, P0y, P0z ), //P0
  new THREE.Vector3( P1x, P1y, P1z ), //P1
  new THREE.Vector3( P2x, P2y, P2z ), //P2
  new THREE.Vector3( P3x, P3y, P3z ), //P3
  new THREE.Vector3( P4x, P4y, P4z ), //P4
  new THREE.Vector3( P5x, P5y, P5z ), //P5
  new THREE.Vector3( P6x, P6y, P6z ), //P6
  new THREE.Vector3( P7x, P7y, P7z ), //P7
] );
```



3. THREE.JS – IMPLEMENTÁCIA KRIVKY V 3D

- **Úloha:** Implementujte vykreslenie spline krivky pomocou *Catmull-Romového algoritmu* pomocou *Three.js*.

1. Na začiatok skriptu „ThreeImport.js“ vložte nasledujúcu implementáciu tak aby premenné boli globálne:

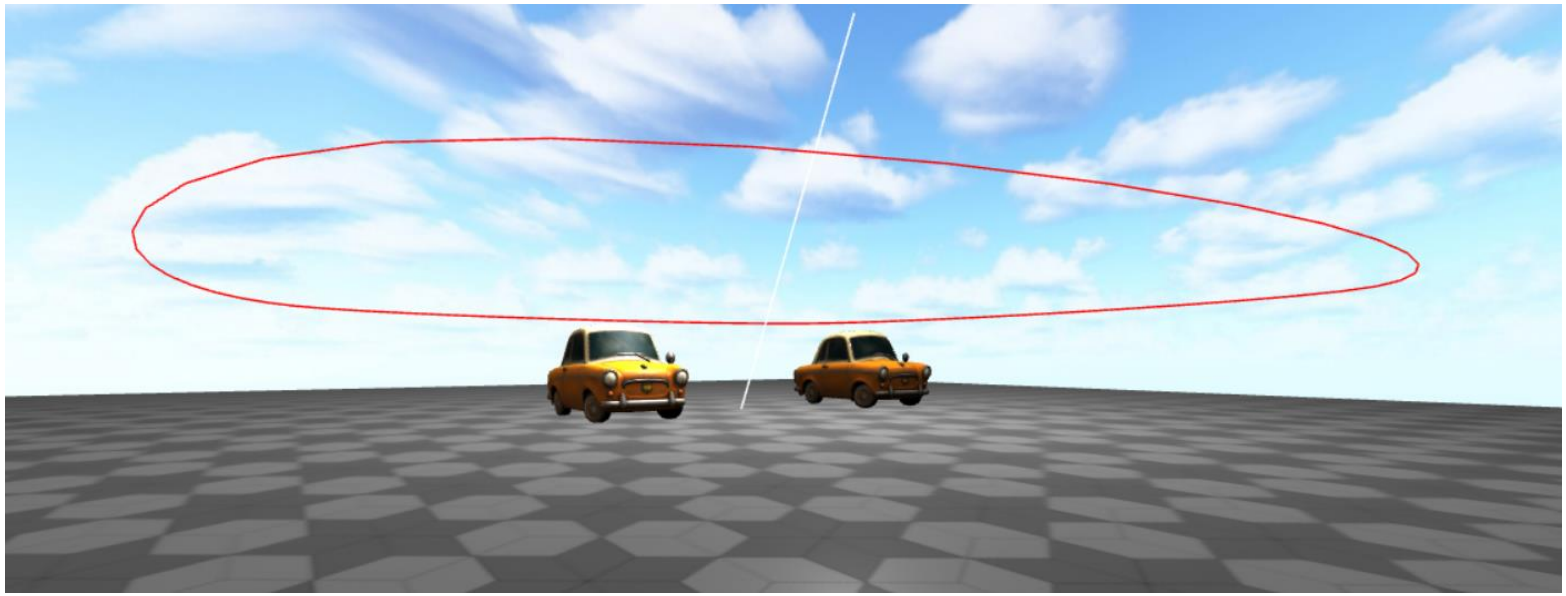
```
var curve = new THREE.CatmullRomCurve3( [
    new THREE.Vector3( -5,1,5 ),
    new THREE.Vector3( 5,1,5 ),
    new THREE.Vector3( 5,1,-5 ),
    new THREE.Vector3( -5,1,-5 ),
], true );
var points = curve.getPoints( 50 );
var geometry = new THREE.BufferGeometry().setFromPoints( points );
var material = new THREE.LineBasicMaterial( { color : 0xff0000 });
var curveObject = new THREE.Line( geometry, material );
var PosIndex = 0;
```

2. Následne v metóde **render()** zobrazte objekt krivky :

```
scene.add(curveObject);
```

3. THREE.JS – IMPLEMENTÁCIA KRIVKY V 3D

3. Overte správnosť implementácie pomocou webového prehliadača.



3. *THREE.JS* – VPLYV POČTU RIADIACICH BODOV NA TVAR KRIVKY

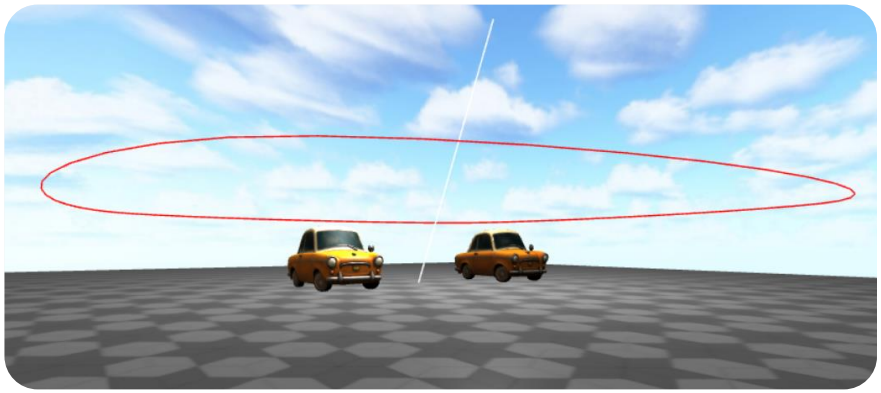
- **Úloha:** Zistite zmenu hladkosti krivky podľa počtu riadiacich bodov.

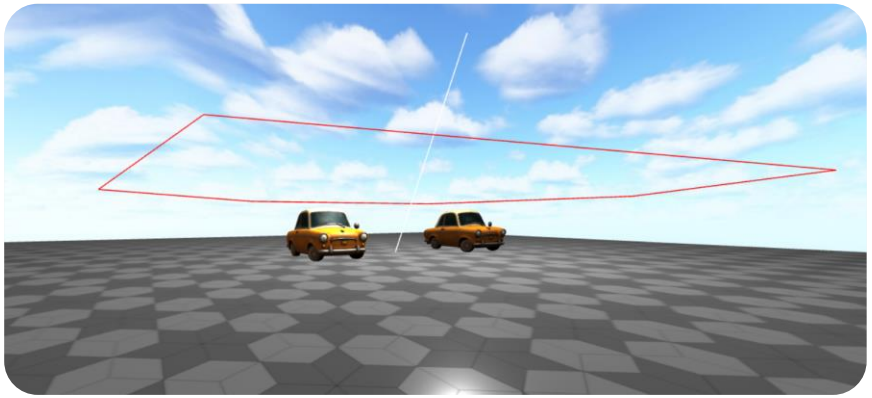
1. Nastavte/zmeňte počet riadiacich bodov na **50** a neskôr na **6** pomocou nasledujúceho skriptu:

```
var points = curve.getPoints( 50 );
```

```
var points = curve.getPoints( 6 );
```

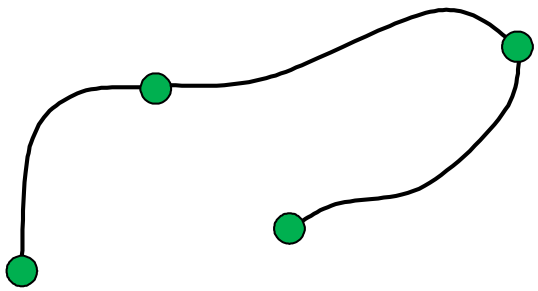
2. Overte správnosť implementácie pomocou webového prehliadača.



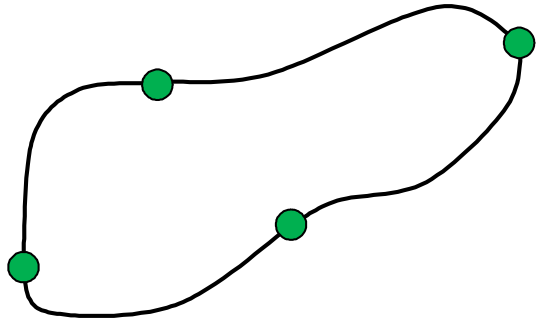


3. *THREE.JS* – CYKLICKÁ A ACYKLICKÁ KRIVKA

- Krivka môže byť uzavretá alebo neuzavretá podľa potreby.
- Ak je **spojený posledný riadiaci bod krivky s prvým pomocou segmentu**, potom je krivka uzavretá a teda **cyklická**.
- **V opačnom prípade**, ak tieto body spojené nie sú, sa jedná o **acyklickú krivku.**



acyklická krivka



cyklická krivka

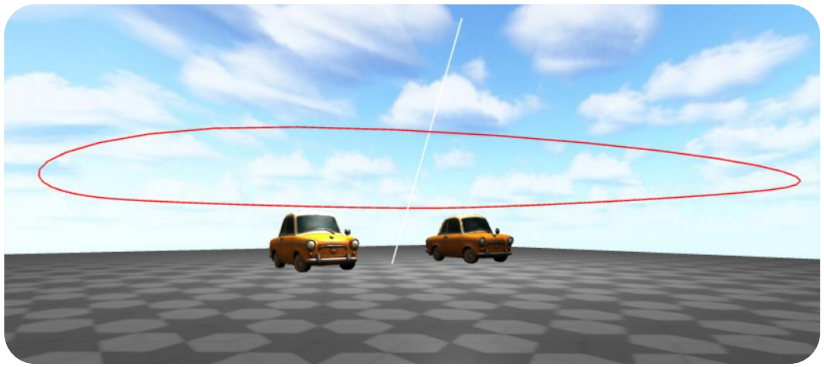
3. THREE.JS – CYKLICKÁ A ACYKLICKÁ KRIVKA

- **Úloha:** Zobrazte krivku najprv ako cyklickú a potom ako acyklickú pri počte riadiacich bodov 50.

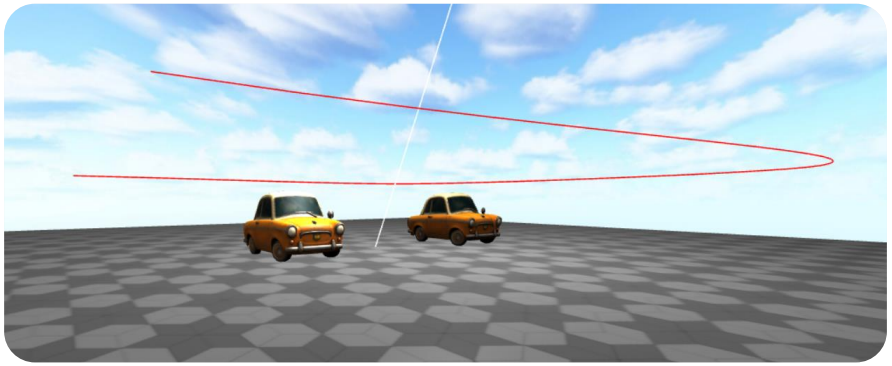
```
var points = curve.getPoints( 50 );
```

1. Na začiatku skriptu „**ThreeImport.js**“ vložte/upravte nasledujúcu implementáciu. Posledným parametrom sa riadi uzavretosť (cyklickosť) krivky (**true** = uzavretá (cyklická) / **false** = otvorená (acyklická).
2. Sledujte zmenu zobrazenia vo webovom prehliadači.

```
var curve = new
THREE.CatmullRomCurve3( [
    new THREE.Vector3( -5,1, 5 ),
    new THREE.Vector3(  5,1, 5 ),
    new THREE.Vector3(  5,1,-5 ),
    new THREE.Vector3( -5,1,-5 ),
    ], true );
```

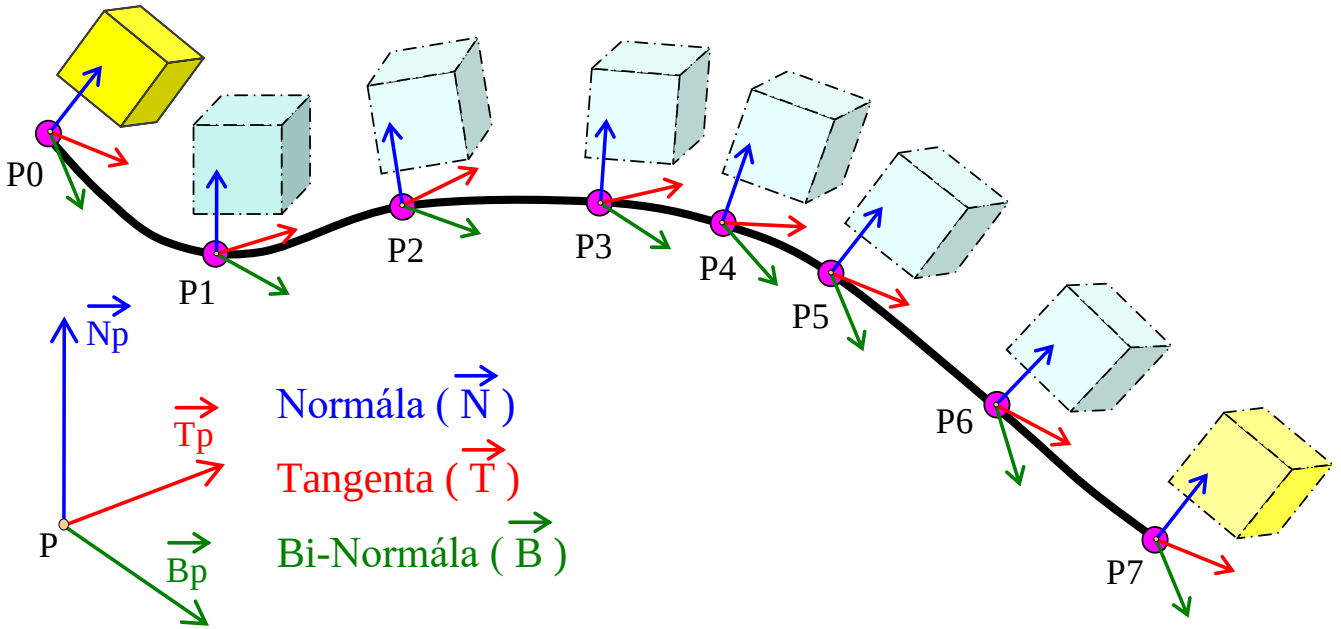


```
var curve = new
THREE.CatmullRomCurve3( [
    new THREE.Vector3( -5,1, 5 ),
    new THREE.Vector3(  5,1, 5 ),
    new THREE.Vector3(  5,1,-5 ),
    new THREE.Vector3( -5,1,-5 ),
    ], false );
```



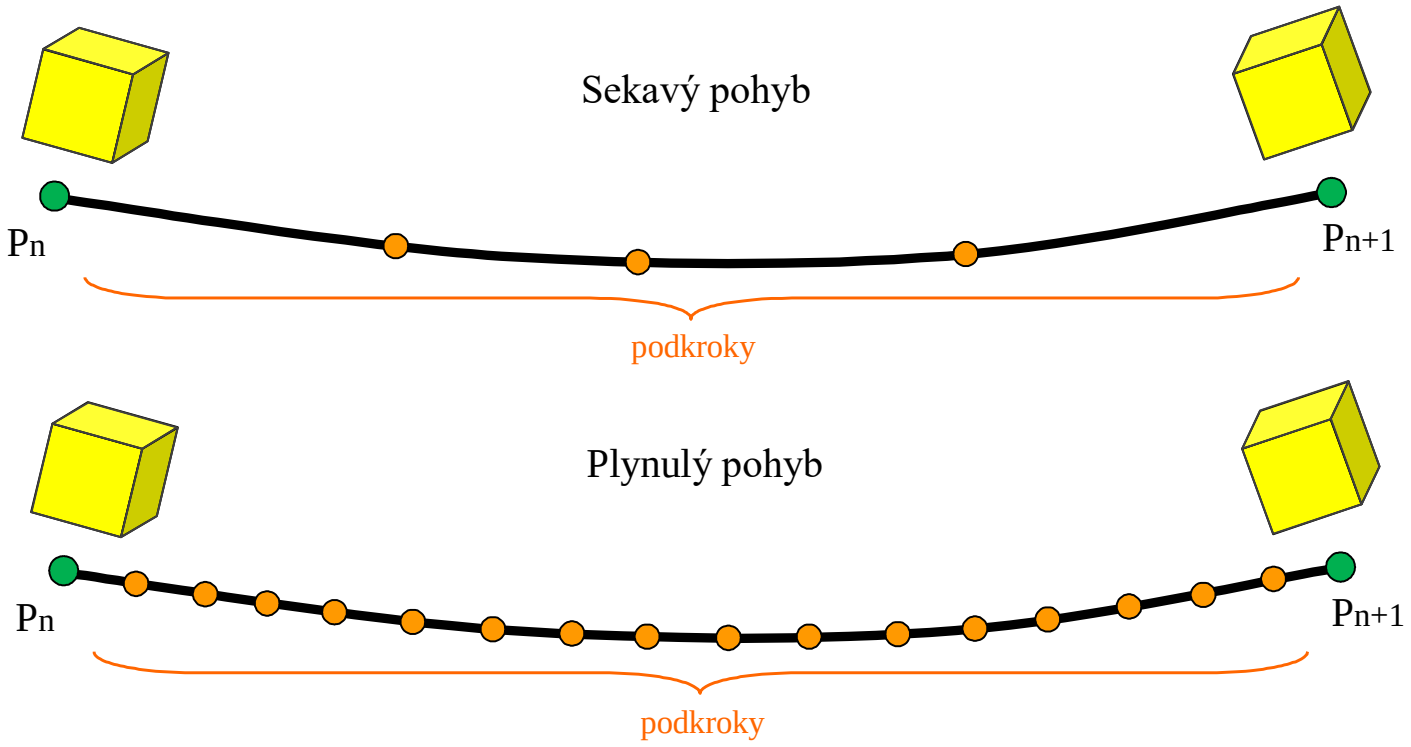
3. THREE.JS – VIAZANIE POHYBU OBJEKTU PO KRIVKE

- Krivku je možné využiť na riadenie trajektórie pohybu objektu napr. pri animácii. Základnú filozofiu ukazuje nasledujúci obrázok.
- Z každého bodu krivky, ktorý je použitý ako riadiaci bod animácie pohybu objektu, sú potom vedené vektory normály, bi-normály a tangenty.



3. THREE.JS – VIAZANIE POHYBU OBJEKTU PO KRIVKE

- Tieto sú veľmi dôležité pre riadenie pohybu objektu po krivke. Plynulosť pohybu objektu je daná počtom podkrokov t.j. počtom použitých bodov krivky.



3. THREE.JS – VIAZANIE POHYBU OBJEKTU PO KRIVKE

- **Úloha:** Implementujte pohyb objektu po definovanej krivke.

1. V skripte „**ThreeImport.js**“ do metódy **addObjects()** vložte nasledujúcu implementáciu objektu kocky:

```
var geometryCube = new THREE.BoxGeometry( 1, 1, 1 );
var cubeTexture = new THREE.ImageUtils.loadTexture(
                                    'texture/lirkis.jpg' );
var materialCube = new THREE.MeshBasicMaterial( {
                    map: cubeTexture,
                    side: THREE.DoubleSide} );
cube = new THREE.Mesh( geometryCube, materialCube );
cube.position.set(0, 0, 0);
scene.add( cube );
```

3. THREE.JS – VIAZANIE POHYBU OBJEKTU PO KRIVKE

- **Úloha:** Implementujte pohyb objektu po definovanej krivke.

2. V skripte „**ThreeImport.js**“ vložte do metódy **render()** nasledujúcu implementáciu pre riadenie pohybu kocky po krivke :

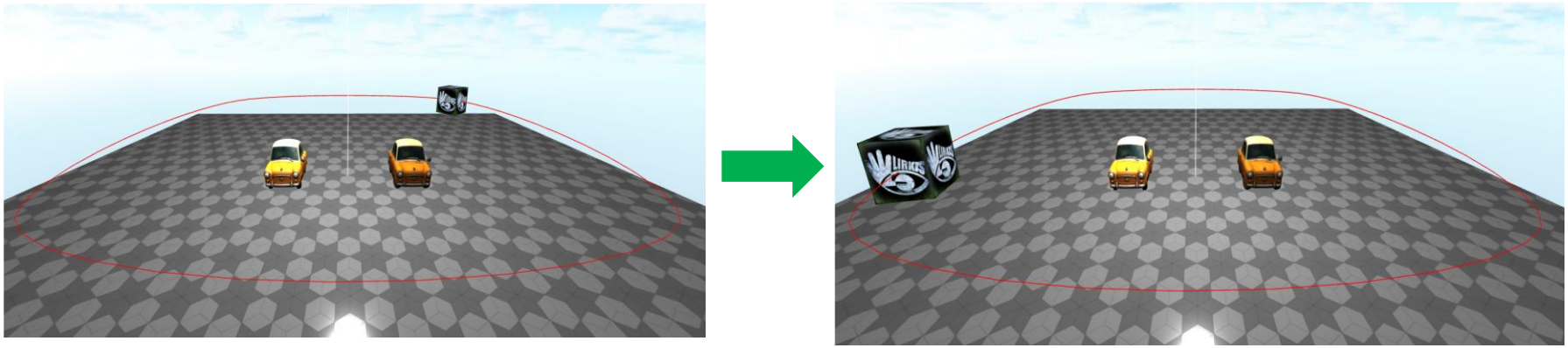
```
PosIndex++;
if (PosIndex > 10000) { PosIndex = 0;}
var camPos = curve.getPoint(PosIndex / 1000);
var camRot = curve.getTangent(PosIndex / 1000);
cube.position.x = camPos.x;
cube.position.y = camPos.y;
cube.position.z = camPos.z;
cube.rotation.x = camRot.x;
cube.rotation.y = camRot.y;
cube.rotation.z = camRot.z;
cube.lookAt(curve.getPoint((PosIndex+1) / 1000));
```

a experimentujte s rotáciou (s/bez *cube.rotation*) a natočením kocky (s/bez *cube.lookAt*)

3. THREE.JS – VIAZANIE POHYBU OBJEKTU PO KRIVKE

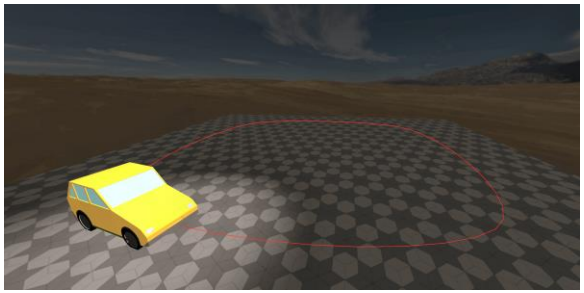
- **Úloha:** Implementujte pohyb objektu po definovanej krivke.

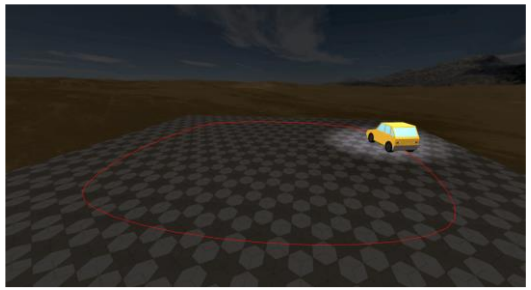
3. Výsledok implementácie overte vo webovom prehliadači.



DOPLŇUJÚCE ÚLOHY

1. Implementujte **pohyb modelu *lirkis_car* po krivke** s jeho sledovaním pomocou už definovaného reflektorového svetla. Upravte svetelné pomery v scéne tak, aby bolo osvetlenie výrazné. Upravte parametre reflektorového svetla, aby nedochádzalo k osvetľovaniu vzdialených objektov, iba modelu (a samozrejme prípadne podlahy).





2. Vyskúšajte implementovať **pohyb kamery po krivke**.
3. Implementujte dva objekty pohybujúce sa po krivke za sebou (vláčik). Ktorý vstupný parameter ovplyvní pozíciu objektov tak, aby nasledovali za sebou ?

ÚLOHY NA SAMOSTATNÉ RIEŠENIE

1. Implementujte ovládanie niektorého z modelov áut.
   Pre túto potrebu bude potrebné doplniť parameter metódy **loadOBJects**() tak, aby referenciu importovaného modelu dosadila za globálnu premennú. Následne je možné nad globálnou premennou importovaného modelu volať metódy pre interaktívnu zmenu pozície, rotácie a pod.
2. Implementujte do úlohy 1 z Doplňujúcich úloh reflektorové svetlo na predok pohybujúceho auta s orientáciou v smere pohybu auta.

Q & A

branislav.sobota@tuke.sk
lenka.bubenkova@tuke.sk

Katedra počítačov a informatiky, FEI TU v Košiciach

© 2024