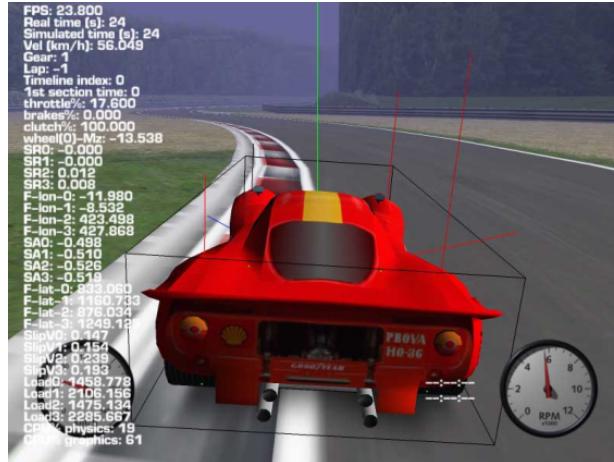


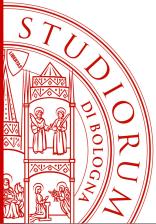
Progetto Car





Progetto Car

- Vogliamo affrontare insieme i passi fondamentali per la realizzazione di un codice WebGL per un simulatore di guida o videogioco tipo “3D car race” o “3D car game”;
- Il passo zero, o progettazione della nostra applicazione Web 3D, consiste nel definire il più precisamente possibile cosa vogliamo che faccia e nel suddividere in passi lo sviluppo del codice per realizzarla;
- Svilupperemo in cinque passi (1-5).



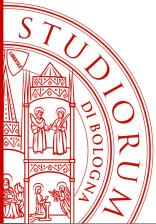
Progetto Car: Passo 0

Progettazione: nella sua essenzialità vogliamo progettare una scena composta da un'automobile su un piano stradale e vogliamo guidare l'auto.

Questa base potrebbe essere utilizzata per differenti videogiochi:

- una corsa di Formula 1,
- una corsa a tempo su un circuito (rally),
- una corsa evitando ostacoli,
- guidare per le strade di una città per raggiungere un luogo,
- per esercitarsi in un parcheggio,
- per una corsa sul ghiaccio,
- ecc.

Oltre agli aspetti di grafica 3D dovremo gestire un minimo di fisica realistica per il movimento dell'auto e se del caso gestire le collisioni con altri oggetti nella scena.



Progetto Car: Passo 0

Passo 1. Il primo passo consiste nel creare la scena in cui navigare con la camera e per iniziare, l'unico oggetto in scena sia l'auto con le ruote; concentriamoci sul problema di far girare le ruote. Progettiamo una semplice geometria che rappresenti un'automobile e le 4 ruote. (Ci serve un codice in cui definire la geometria prevista e mettere in atto le trasformazioni geometriche necessarie; consideriamo da subito una rappresentazione prospettica ed un controllo camera con il mouse).

Passo 2. Il secondo passo consiste nel gestire il movimento dell'auto (avanti e indietro con una certa velocità) con rotazione e sterzo delle ruote, basandoci su un po' di fisica elementare. Definiamo un piano su cui far muovere l'auto. Per semplicità gestiamo gli eventi da tastiera per guidare l'auto: avanti, sinistra, destra, indietro, i classici tasti WASD

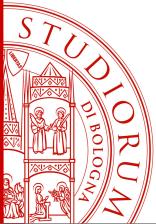


Progetto Car: Passo 0

Passo 3. Nel terzo passo introduciamo un modello di illuminazione che preveda una luce direzionale, gestione di opportuni materiali, ecc.. Ci concentriamo anche su una corretta gestione dei Frame per Secondo (fps) e/o tempo di simulazione della fisica introdotta.

Passo 4. Nel quarto passo introduciamo una geometria mesh per l'auto e le ruote e potremmo aggiungere differenti controlli del movimento. (Ci servono le nozioni su mesh, strutture dati, formati mesh ed elementi di modellazione).

Passo 5. Nel quinto passo aggiungiamo alcuni effetti grafici per rendere il tutto più realistico, ma cercando di non penalizzare il real-time. (Ci servono le nozioni sul rendering con l'introduzione di materiali, texture ed altro ancora come ombre, ecc.).



Passo 1

Adattiamo un codice già visto che definita una geometria la visualizzi in proiezione prospettica e permetta un movimento camera nella scena.

Vogliamo controllare la camera con il mouse.

Usiamo come impianto iniziale di codice uno di quelli visti in WebGL per disegnare un cubo e ruotarlo con il mouse:

codice in [HTML5_webgl_2](#)
[cube_interact_shaderscript.html](#) e [.js](#)

dupliciamolo e rinominiamolo:

[cube_interact2.html](#) e [.js](#)



Passo 1

Modifichiamo questo codice in modo che:

- 1.il cubo venga visualizzato con le facce tutte dello stesso colore; ora sono di colore diverso; perché siano dello stesso colore cosa dobbiamo fare?
- 2.se le facce sono dello stesso colore per distinguerle si disegnino anche i lati; cosa dobbiamo fare?
- 3.si generi una copia traslata e ruotata del cubo, in modo da averne due in scena; come si fa?
- 4.si preveda un controllo per ruotare interattivamente/animare solo il nuovo cubo, lasciando fermo il primo; come?





Passo 1

Affrontiamo il punto 3.

```
//Copia ruotata e poi traslata del cubo  
mo_matrix2=m4.identity();  
mo_matrix2=m4.translate(mo_matrix2, 0, 0, 2.2);  
mo_matrix2=m4.zRotate(mo_matrix2, degToRad(alpha));  
mo_matrix2=m4.scale(mo_matrix2, 0.75, 0.75, 0.75);  
gl.uniformMatrix4fv(_Mmatrix, false, mo_matrix2);
```

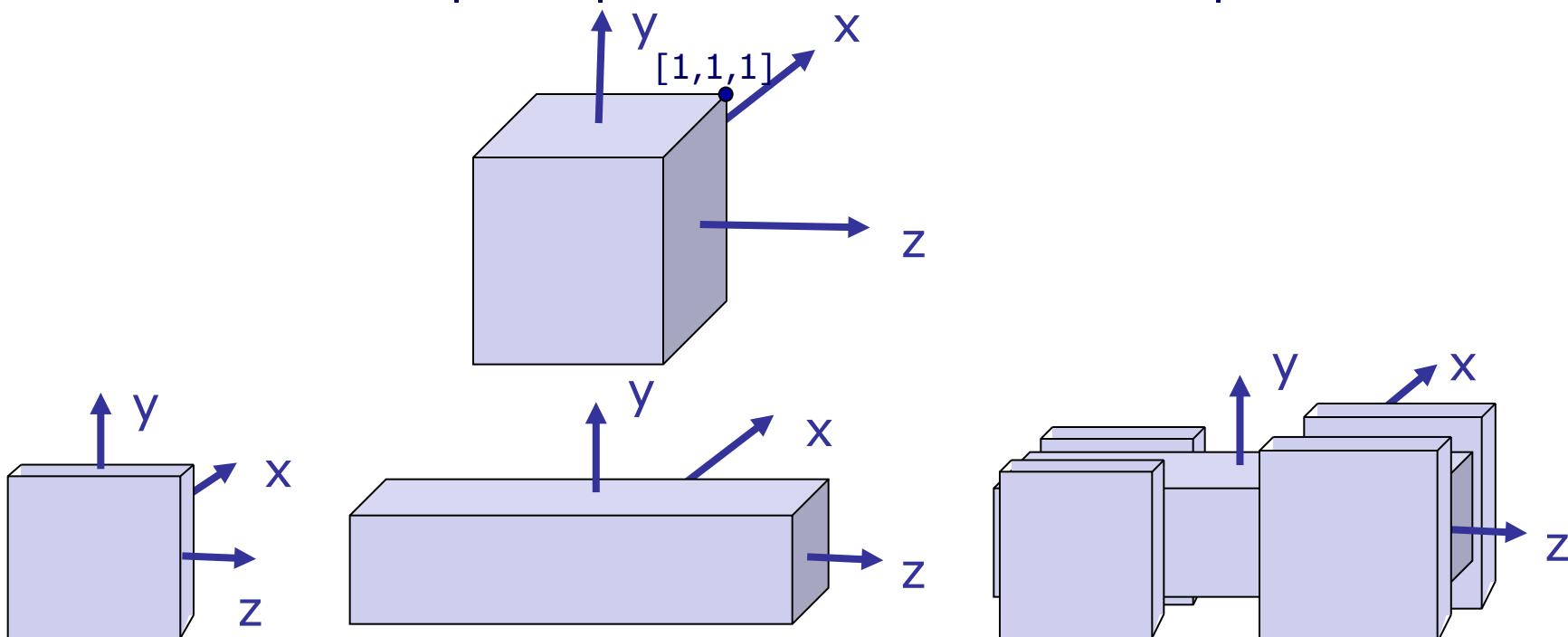
Attenzione: se si vuole prima ruotare e poi traslare, si dovrà prima chiamare la funzione che trasla e dopo quella che ruota!!

```
mo_matrix2=I  
mo_matrix2=mo_matrix2*T=I*T=T  
mo_matrix2=mo_matrix2*R=T*R  
mo_matrix2=mo_matrix2*S=T*R*S
```

Regola mnemonica: l'ultima chiamata è la prima applicata.

Passo 1

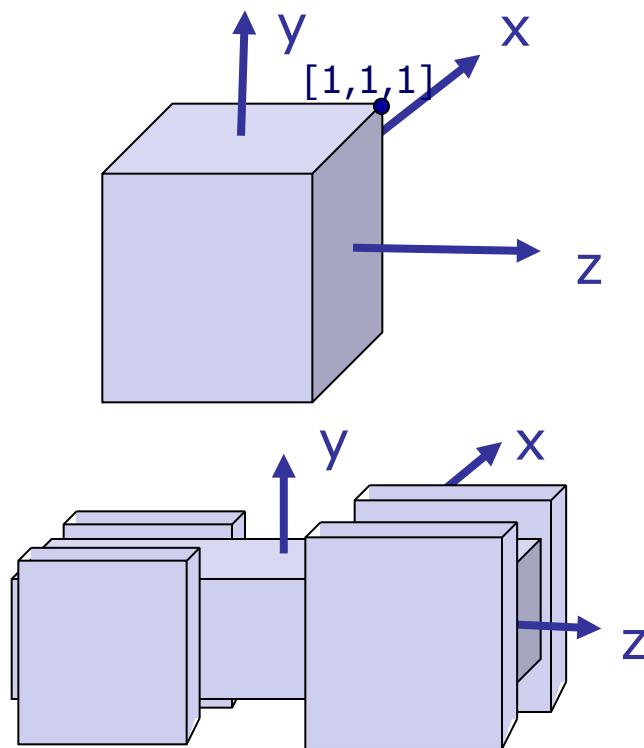
Sfruttando quanto visto decidiamo di usare come semplice forma per l'auto 5 parallelepipedi (uno per la carrozzeria e 4 per le ruote); usiamo la sola geometria cubo che scaleremo e trasferremo nello spazio per dare forma alle varie parti.



codice `cg_car1.html` e `car1.js`

Passo 1

Definiamo il cubo di centro l'origine e vertici $[\pm 1, \pm 1, \pm 1]$; quindi applichiamo delle traslazioni e scale, per modellare le varie parti. Definiamo una function `drawCube()` per disegnare ogni parte.

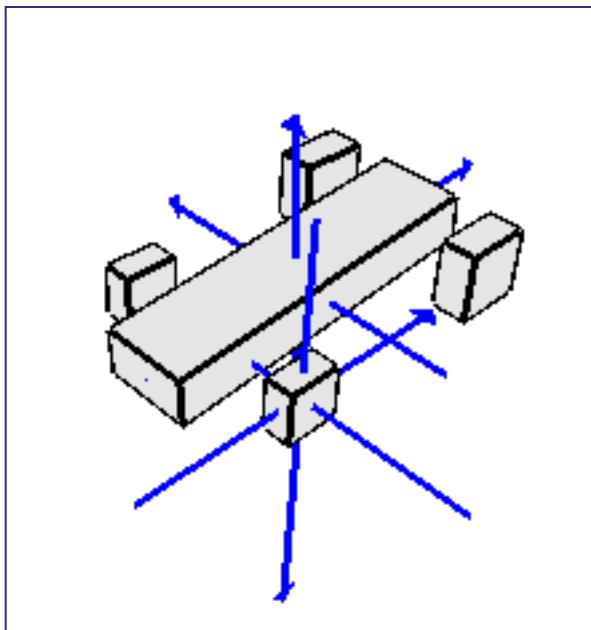


```
// carrozzeria
Scalef(0.25 , 0.14 , 1);
drawCube();
// ruota posteriore D
Translatef( 0.58,-0.05,+0.8);
Scalef(0.1, 0.20, 0.20);
drawCube();
// ruota posteriore S
Translatef(-0.58,-0.05,+0.8);
Scalef(0.1, 0.20, 0.20);
drawCube();
// ruota anteriore D
Translatef( 0.58,-0.05,-0.55);
Scalef(0.08, 0.15, 0.15);
drawCube();
// ruota anteriore S
Translatef(-0.58,-0.05,-0.55);
Scalef(0.08, 0.15, 0.15);
drawCube();
```

Passo 1

Concentriamoci sulle ruote;
vogliamo farle ruotare.

```
//inizializziamo mozzo  
var mozzo=0  
var mozzo_step=degToRad(15);
```



```
if (!drag)  
mozzo-=mozzo_step;
```

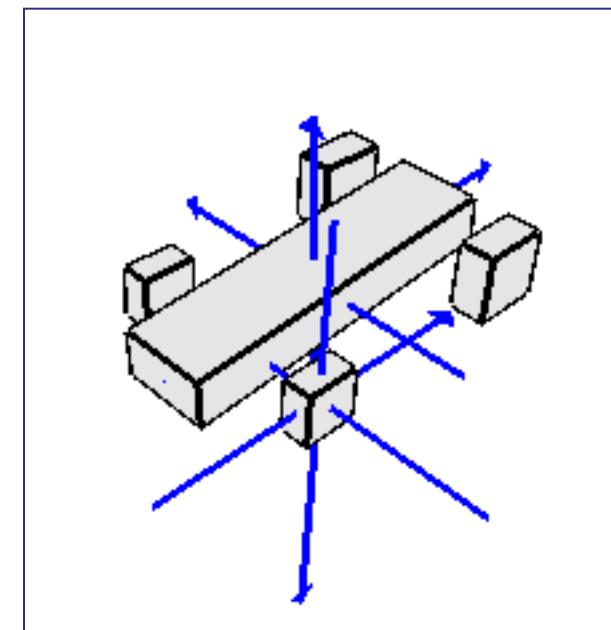
```
// carrozzeria  
Scalef(0.25 , 0.14 , 1);  
drawCube();  
// ruota posteriore D  
Translatef( 0.58,-0.05,+0.8);  
Rotatef(mozzo,1,0,0);  
Scalef(0.1, 0.20, 0.20);  
drawCube();  
// ruota posteriore S  
Translatef(-0.58,-0.05,+0.8);  
Rotatef(mozzo,1,0,0);  
Scalef(0.1, 0.20, 0.20);  
drawCube();  
// ruota anteriore D  
Translatef( 0.58,-0.05,-0.55);  
Rotatef(mozzo,1,0,0);  
Scalef(0.08, 0.15, 0.15);  
drawCube();  
// ruota anteriore S  
Translatef(-0.58,-0.05,-0.55);  
Rotatef(mozzo,1,0,0);  
Scalef(0.08, 0.15, 0.15);  
drawCube();
```

Passo 2

Usando questa semplice geometria per l'auto e le ruote, vogliamo introdurre il movimento seguendo un moto fisico corretto, determinato da una velocità con tanto di rotolamento e sterzo ruote (vorremo controllare da tastiera questi due parametri).

Sviluppiamo un codice JavaScript in cui definiamo oltre all'auto un piano su cui farla muovere e alcune funzioni di controllo del movimento.

Per simulare correttamente la fisica faremo uso della gestione del tempo per ridisegnare i frame.



codice cg_car2.html e car2.js

Passo 2: la fisica

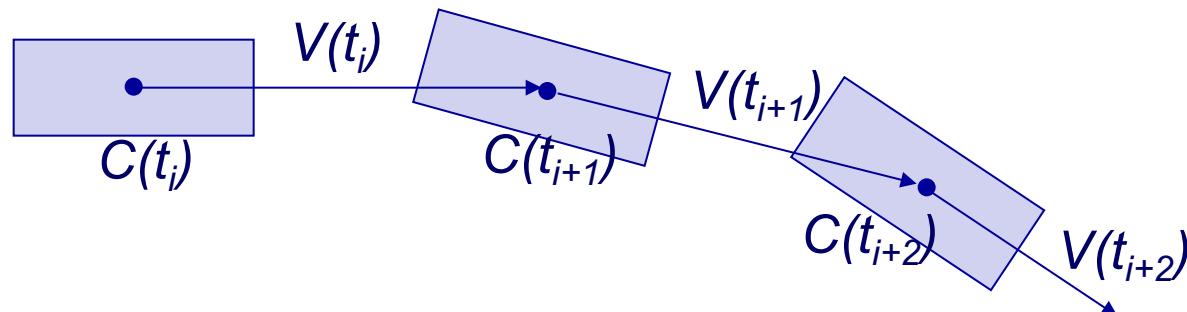
Faremo uso dell'equazione del moto uniformemente accelerato

$$C(t) = c_0 + v_0 t + \frac{1}{2} a t^2$$

ed in particolare simuleremo questo moto a istanti di tempo unitari; allora se chiamiamo $t_i = i$ con $i=0,1,\dots$ gli istanti di tempo nei quali vorremo calcolare e rappresentare la posizione dell'auto, questa sarà data da:

$$C(i+1) = C(i) + V(i) + \frac{1}{2} a \quad \text{con} \quad V(i) = V(i-1) + a \quad (V(t) = v_0 + a t)$$

Ad ogni istante $t_i=i$ avremo un vettore velocità, che sarà definito completamente conoscendo **direzione, verso e modulo**; il **modulo** sarà determinato da un aumento o diminuzione costante nel tempo, la **direzione** dall'angolo di rotazione richiesto per le ruote anteriori, il **verso** sarà positivo o negativo rispetto al fronte dell'auto.

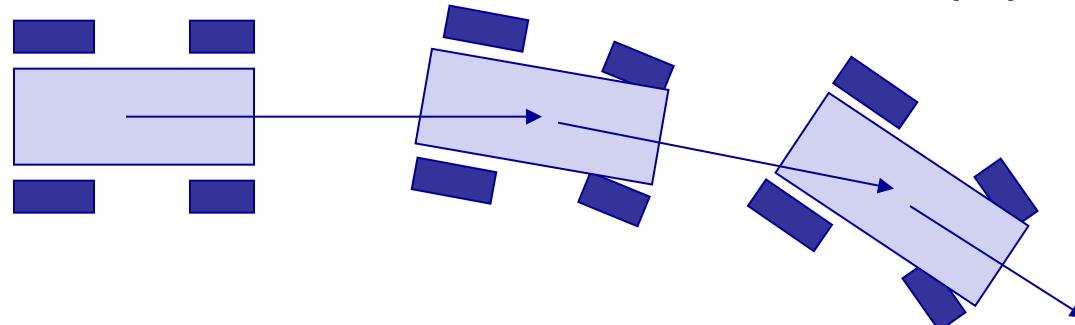


Passo 2: la fisica

Determinata la posizione dell'auto $C(i+1)$, per l'istante successivo $i+1$ dovremo applicare le opportune trasformazioni geometriche per ottenere la geometria in quella posizione (rotazione e traslazione);

Ma vogliamo rappresentare correttamente anche lo sterzo richiesto per le ruote anteriori insieme alla rotazione corretta sul proprio asse rispetto alla velocità dell'auto .

```
sterzo = 0;  
velSterzo = 3.4;           // A  
velRitornoSterzo = 0.93; // B, sterzo massimo = A*B/(1-B) = 45.17 gradi  
  
//gestione dello sterzo  
sterzo ±= velSterzo;    // + sterzo a sinistra, - a destra  
sterzo *= velRitornoSterzo; // ritorno a volante fermo; (...(A*B+A)*B+..+A)*B
```





Passo 2: la fisica

$$\lim_{n \rightarrow \infty} (\dots (AB + A)B + A)B + \dots + A)B =$$

$$\lim_{n \rightarrow \infty} (AB + AB^2 + AB^3 + \dots + AB^n) =$$

$$\lim_{n \rightarrow \infty} AB(1 + B + \dots B^{n-1}) =$$

$$\lim_{n \rightarrow \infty} \frac{AB}{(1 - B)} (1 + B + \dots B^{n-1})(1 - B) =$$

$$\lim_{n \rightarrow \infty} \frac{AB}{(1 - B)} (1 + \cancel{B} + \dots + \cancel{B}^{n-1} - \cancel{B} - \cancel{B}^2 - \dots - \cancel{B}^{n-1} - B^n) =$$

$$\lim_{n \rightarrow \infty} \frac{AB}{(1 - B)} (1 - B^n) = \frac{AB}{(1 - B)}$$

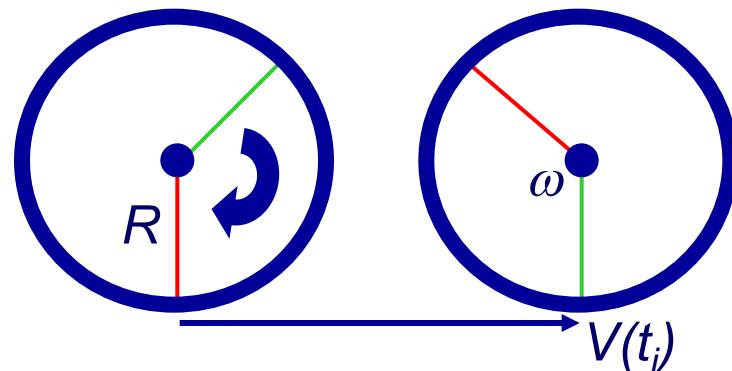
$$\lim_{n \rightarrow \infty} B^n = 0 \quad \text{se } 0 < B < 1$$

Passo 2: la fisica

Dalla velocità V dell'auto vogliamo determinare il corretto rotolamento delle ruote; la velocità angolare ω è data dalla formula:

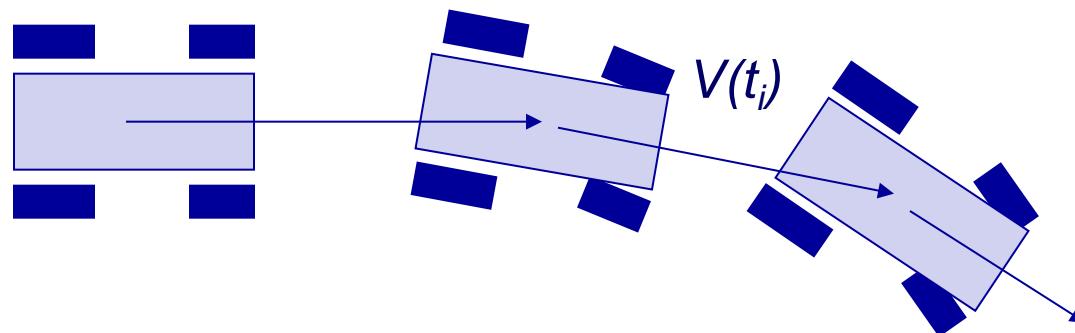
$$\omega = V / R$$

con R raggio della ruota.



$$\alpha = \frac{180}{\pi} \frac{V}{R}$$

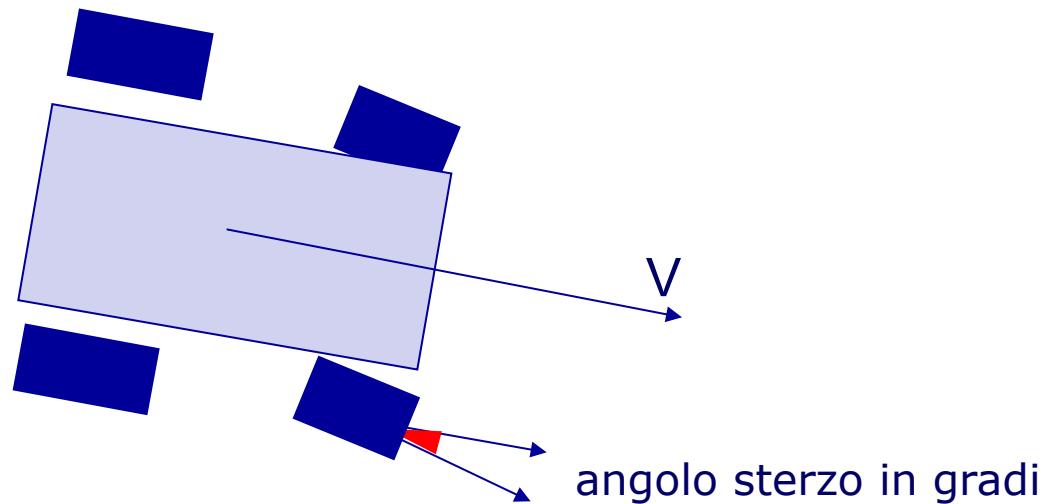
Angolo in gradi



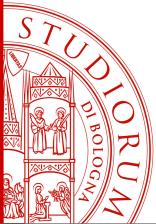
Passo 2: la fisica

L'orientamento della macchina segue quello dello sterzo a seconda del modulo della velocità, secondo il seguente semplice modello:

$$\text{facing} = (|V| * \text{grip}) * (\text{angolo sterzo})$$



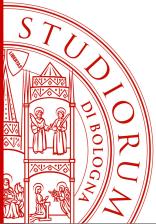
Si gestisce anche l'attrito, una velocità massima e il grip (aderenza delle gomme al terreno).



Passo 2: la fisica

```
function CarInit(){  
    // inizializzo lo stato della macchina  
    px=py=pz=facing=0; // posizione e orientamento  
    mozzoA=mozzoP=sterzo=0; // stato  
    vx=vy=vz=0; // velocita' attuale  
    // inizializzo la struttura di controllo  
    key=[false,false,false,false];  
    velSterzo=3.4; // A  
    velRitornoSterzo=0.93; // B, sterzo massimo = A*B / (1-B)  
    accMax = 0.0055;  
  
    // attriti: percentuale di velocita' che viene mantenuta  
    // 1 = no attrito  
    // <<1 = attrito grande  
    attritoZ = 0.991; // piccolo attrito sulla Z (nel senso di rotolamento delle ruote)  
    attritoX = 0.8; // grande attrito sulla X (per non fare slittare la macchina)  
    attritoY = 1.0; // attrito sulla y  
  
    // Nota: velocita' max = accMax * attritoZ / (1 - attritoZ)  
  
    raggioRuotaA = 0.25;  
    raggioRuotaP = 0.35;  
    grip = 0.45; // quanto il facing macchina si adegua velocemente allo sterzo  
}
```

oggetto Car



Passo 2: la fisica

```
// DoStep: facciamo un passo di fisica (a delta-t costante)
function CarDoStep(){
    // computiamo l'evolversi della macchina
    var vxm, vym, vzm; // velocita' in spazio macchina

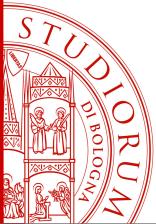
    // da vel frame mondo a vel frame macchina
    var cosf = Math.cos(facing*M_PI/180.0);
    var sinf = Math.sin(facing*M_PI/180.0);
    vxm = +cosf*vx - sinf*vz;
    vym = vy;
    vzm = +sinf*vx + cosf*vz;

    // gestione dello sterzo
    if (key[1]) sterzo+=velSterzo;
    if (key[3]) sterzo-=velSterzo;
    sterzo*=velRitornoSterzo; // ritorno a volante fermo

    if (key[0]) vzm-=accMax; // accelerazione in avanti
    if (key[2]) vzm+=accMax; // accelerazione indietro

    // attriti (semplificando)
    vxm*=attritoX;
    vym*=attritoY;
    vzm*=attritoZ; ...
```

metodo CarDoStep



Passo 2: la fisica

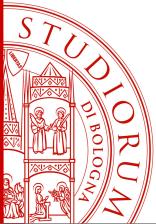
```
...
// l'orientamento della macchina segue quello dello sterzo
//(a seconda della velocita' sulla z)
facing = facing - (vzm * grip) * sterzo;

// rotazione mozzo ruote (a seconda della velocita' sulla z)
var da ; //delta angolo
da = (180.0 * vzm) / (M_PI * raggioRuotaA);
mozzoA += da;
da = (180.0 * vzm) / (M_PI * raggioRuotaP);
mozzoP += da;

// ritorno a vel coord. mondo
vx = + cosf * vxm + sinf * vzm;
vy = vym;
vz = - sinf * vxm + cosf * vzm;

// posizione = posizione + velocita * delta_t (ma e' delta_t costante=1)
px += vx;
py += vy;
pz += vz;
}
```

metodo CarDoStep



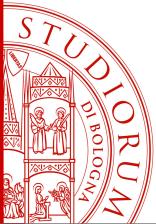
Passo 2: la fisica

Gestiamo il tempo per ridisegnare i frame e per simulare correttamente la fisica.

Primo metodo basato su FPS:

```
const FRAMES_PER_SECOND = 30; // Valid values are 60,30,20,15,10...
// set the mim time to render the next frame
const FRAME_MIN_TIME = (1000/60) * (60 / FRAMES_PER_SECOND) - (1000/60) * 0.5;
var lastFrameTime = 0; // the last frame time
function update(time){
    if(time-lastFrameTime < FRAME_MIN_TIME){ //skip the frame if the call is too early
        CarDoStep();
        window.requestAnimationFrame(update);
        return; // return as there is nothing to do
    }
    lastFrameTime = time; // remember the time of the rendered frame
    render(); //render the frame
    window.requestAnimationFrame(update); // get next frame
}
CarInit();
update(); // start animation
window.requestAnimationFrame(update);
```

main loop



Passo 2: la fisica

Secondo metodo basato su tempo di simulazione:

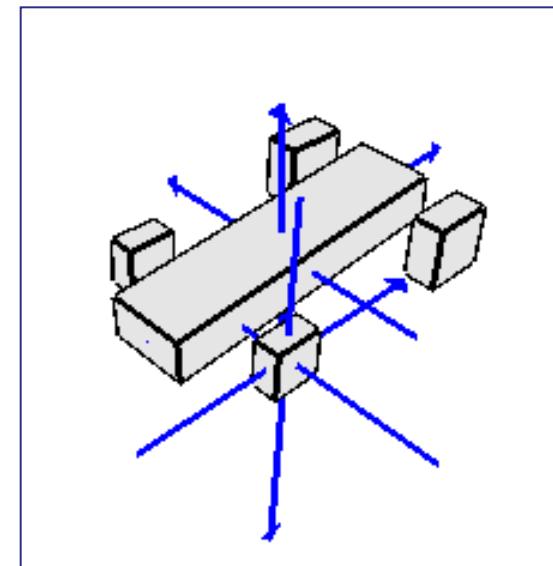
```
var doneSomething=false;
var nstep=0;
const PHYS_SAMPLING_STEP=10; // numero di millisec che un passo di fisica simula
var timeNow=0;
function update(time){
    if(nstep*PHYS_SAMPLING_STEP <= timeNow){ //skip the frame if the call is too early
        CarDoStep();
        nstep++;
        doneSomething=true;
        window.requestAnimationFrame(update);
        return; // return as there is nothing to do
    }
    timeNow=time;
    if (doneSomething) {
        render(); // render the frame
        doneSomthing=false;
    }
    window.requestAnimationFrame(update); // get next frame
}
CarInit();
update(); // start animation
window.requestAnimationFrame(update);
```

main loop

Passo 3

In questa versione introduciamo l'illuminazione e la definizione di materiali;

```
var lightPosition = [2, 7.0, 0, 0.0 ];  
  
var lightAmbient = [0.2, 0.2, 0.2, 1.0 ];  
var lightDiffuse = [1.0, 1.0, 1.0, 1.0 ];  
var lightSpecular = [1.0, 1.0, 1.0, 1.0 ];  
  
var materialAmbient = [1.0, 0.0, 1.0, 1.0];  
var materialDiffuse = [1.0, 0.8, 0.0, 1.0];  
var materialSpecular = [1.0, 0.8, 0.0, 1.0];  
  
var materialShininess = 10.0;
```



in questo caso il colore predominante sarà dato dall'array **materialDiffuse** combinato con **lightDiffuse**

[codice cg_car3.html e car3.js](#)

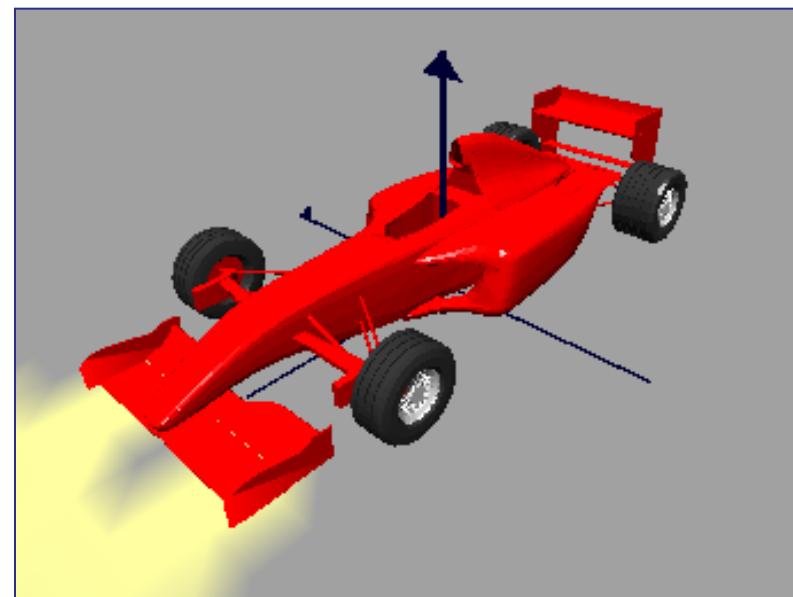
Passo 4

Definiamo un oggetto Mesh che ha fra i suoi metodi il caricamento di mesh in formato obj, oltre che il metodo Render per visualizzarla.

Viene utilizzato un modello 3D di una Ferrari costituito dalla carrozzeria dell'auto, da due ruote (quella davanti e quella dietro) e da due cerchioni in lega.

Verranno istanziati:

carlinga,
wheelBR1,
wheelFR1,
wheelBR2,
wheelFR2.

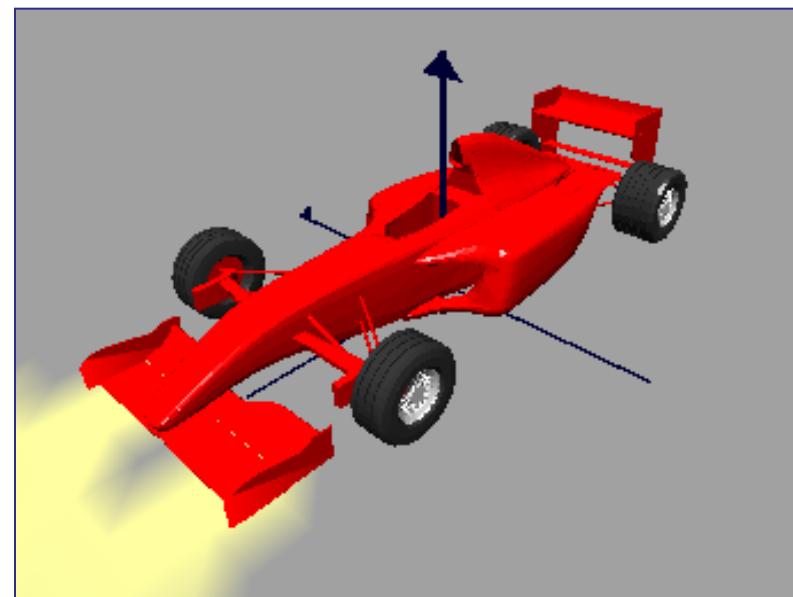


Passo 4

In questo passo utilizziamo l'illuminazione inserita nel passo precedente: una luce direzionale (all'infinito come il sole) e degli attributi per il materiale (altamente riflettente);

Viene utilizzato un colore per il materiale settando il rosso Ferrari;

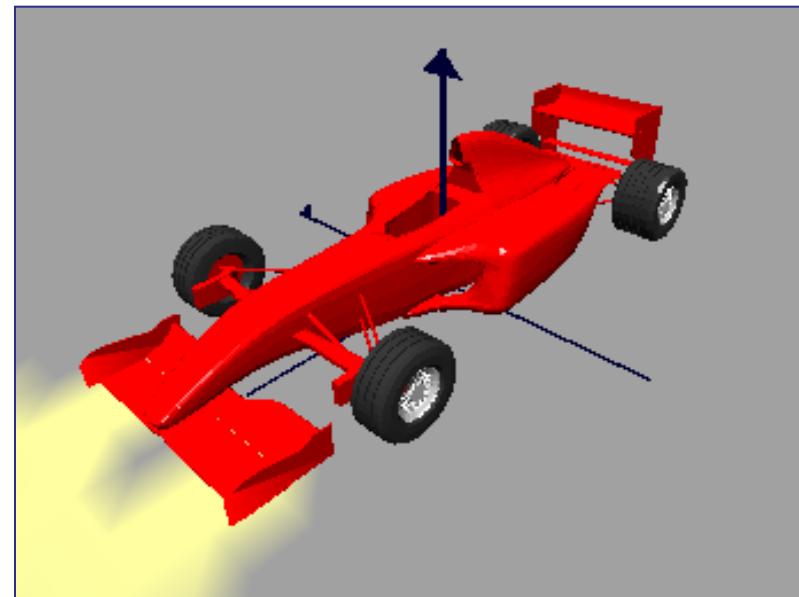
Vengono poi introdotte due luci spot come fari anteriori per l'auto (questo comporta, oltre varie definizioni per le luci di definire una geometria opportuna per il piano stradale).



Passo 4

Si potrebbe gestire l'input anche con un gamepad (questo permeterebbe di variare la velocità e lo sterzo in modo graduale).

Si potrebbe gestire la visualizzazione grafica di fps (frame per secondo) mediante il disegno di una barra colorata.



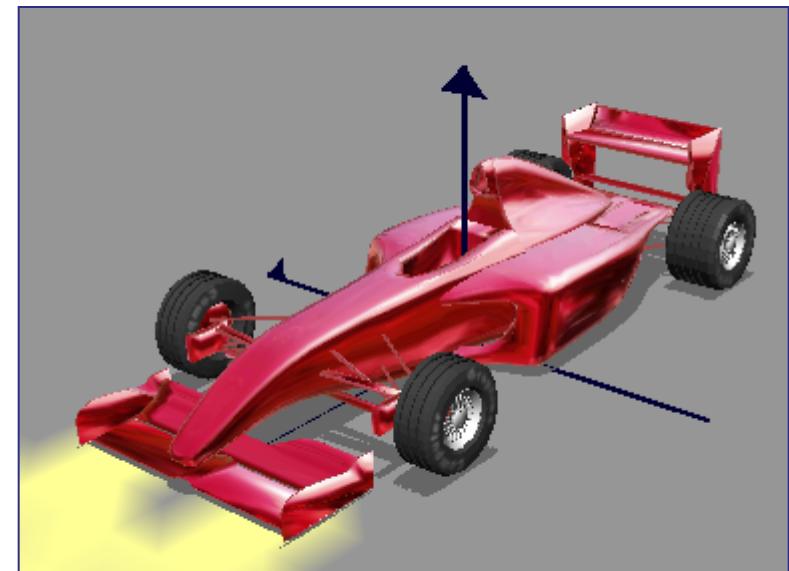
Passo 5

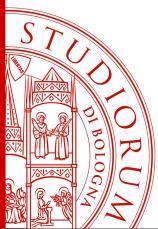
Si vogliono ora aggiungere effetti grafici che aumentino il realismo, ma senza penalizzare il real-time dell'esecuzione.

Si possono introdurre delle texture:

- 1.environment mapping dell'auto;
- 2.projection di una scritta sulle gomme;
- 3.environment mapping/sky box del cielo (texture cielo per creare una scena realistica ambientata all'aperto).

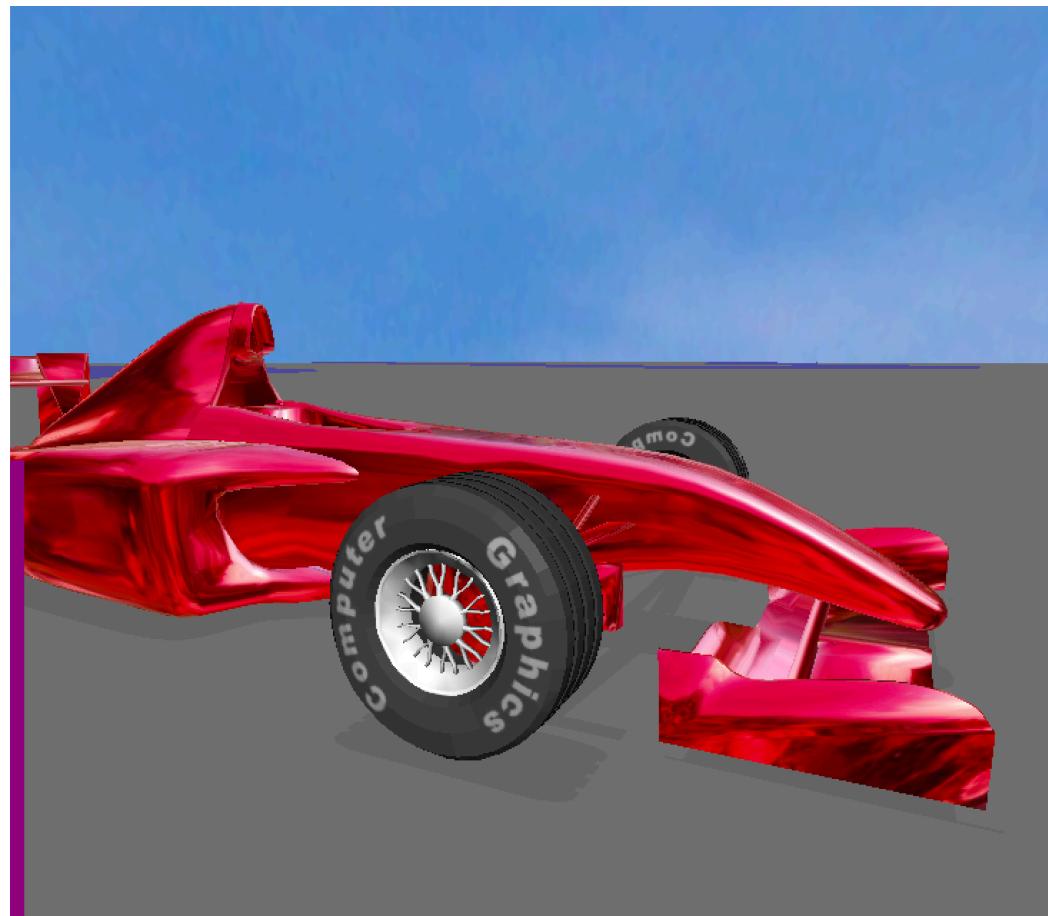
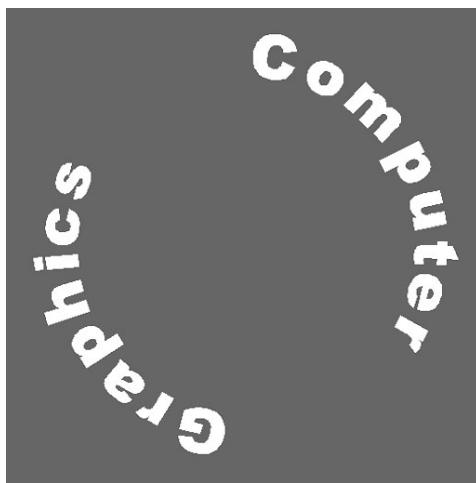
Si possono introdurre le ombre ed in particolare l'ombra dell'auto e delle ruote sull'asfalto.





Passo 5

Un esempio di texture da utilizzare sulle gomme dell'auto:



Passo 5

Si possono gestire più Camere che permettono di seguire l'auto da diverse posizioni fra cui quella del pilota.

Visualizziamo anche in wireframe e progettiamo di abilitare/disabilitare le seguenti possibilità (questo è utile anche ai fini del debug del codice):

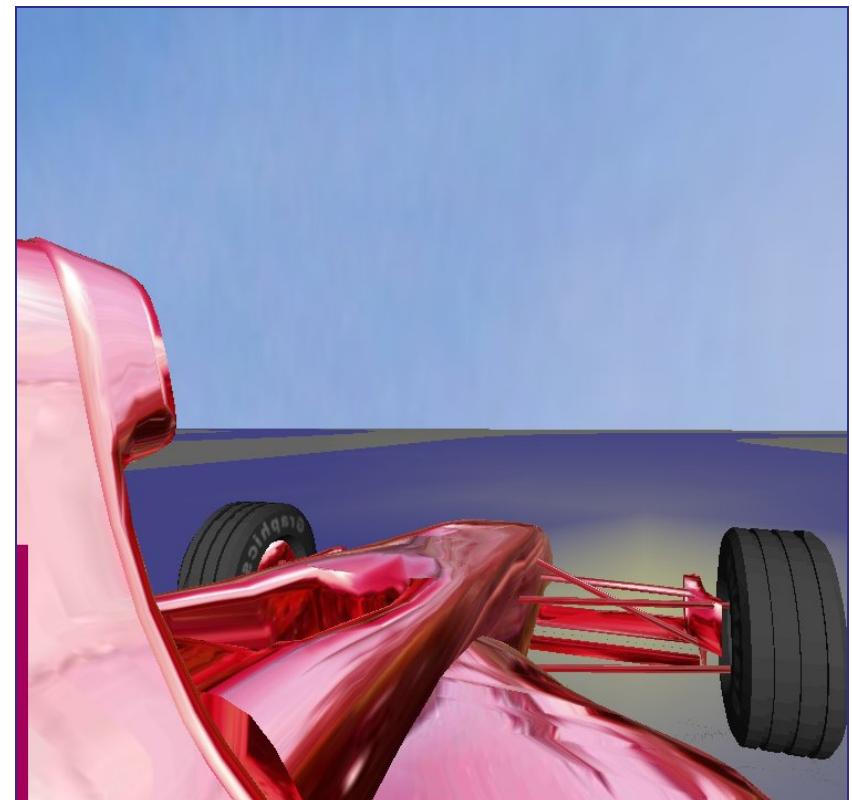
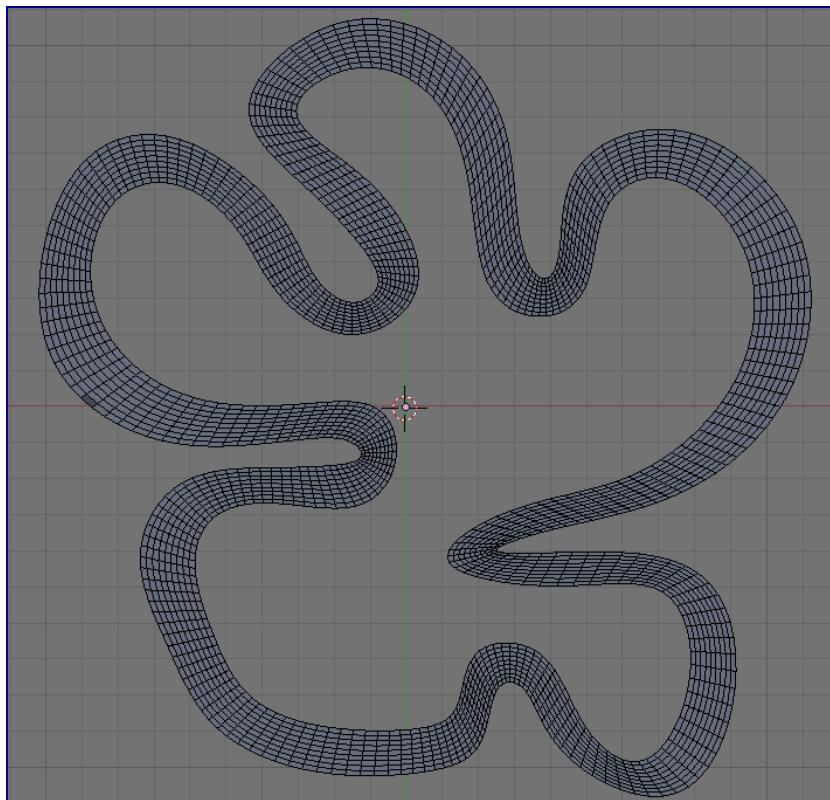
1. differenti camere
2. tipi di rendering
3. texturing/materiali/colori
4. fari
5. ombre
6. ecc.

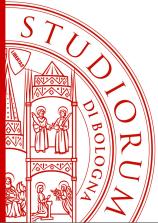


Passo 5

Si può aggiungere una pista (circuito chiuso)

Si potrebbe prevedere una fase demo in cui l'auto viene guidata da un pilota automatico (percorso predefinito dell'auto)





Demo

Vediamo una Web-App di esempio e alcuni progetti sviluppati da vostri colleghi nel passato



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Giulio Casciola
Dip. di Matematica
[giulio.casciola at unibo.it](mailto:giulio.casciola@unibo.it)