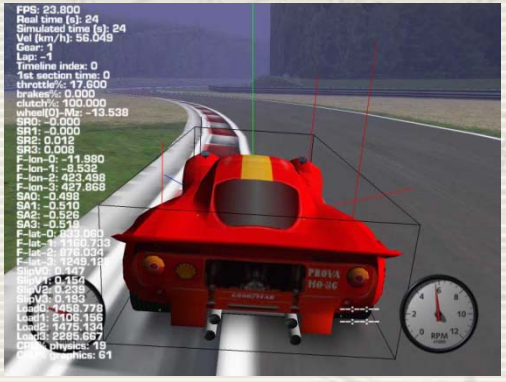
Progetto Car (OpengL/SDL)



Progetto Car

- Vogliamo affrontare insieme i passi fondamentali per la realizzazione di un codice OpenGL/SDL di grafica 3D real-time per un un simulatore di guida od anche un videogioco tipo “3D car race” o “3D car game”.
- Procederemo in cinque passi (0-4), i primi tre dei quali li affronteremo in questa lezione, gli altri due nelle prossime.
- Il passo zero, o progettazione del videogioco, consiste nel definire il più precisamente possibile cosa vogliamo fare e nel suddividere in passi (1-4) lo sviluppo del codice.

Vogliamo progettare una scena composta da un piano stradale, almeno un’automobile, ed altro a piacere; vogliamo far muovere l’auto in tempo reale (guidarla) con un minimo di fisica realistica.

Progetto Car: Passo 0

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; sempre per iniziare concentriamo su problema di far ruotare le ruote. Progettiamo una semplice geometria che rappresenti un'automobile e le 4 ruote.

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 (usiamo per semplicità la tastiera).

3. Nel terzo passo introduciamo una geometria mesh per l'auto e le ruote e aggiungiamo il controllo del movimento via joypad. (Dovremo acquisire le nozioni necessarie su mesh, strutture dati, formati mesh, elementi di modellazione).

4. Nel quarto passo aggiungiamo alcuni effetti grafici per rendere il tutto più realistico, ma cercando di non penalizzare il real-time. (Dovremo acquisire le nozioni di rendering con algoritmi di shading, applicazione di modelli di illuminazione, texturing, ombre ed altro ancora).

Passo 1

Adottiamo un codice già visto che permetta un movimento camera nella scena.

Vogliamo controllare la camera con il motion del mouse.

Usiamo come impianto iniziale di codice quello visto per OpenGL/SDL con un ciclo degli eventi e una funzione di disegno.

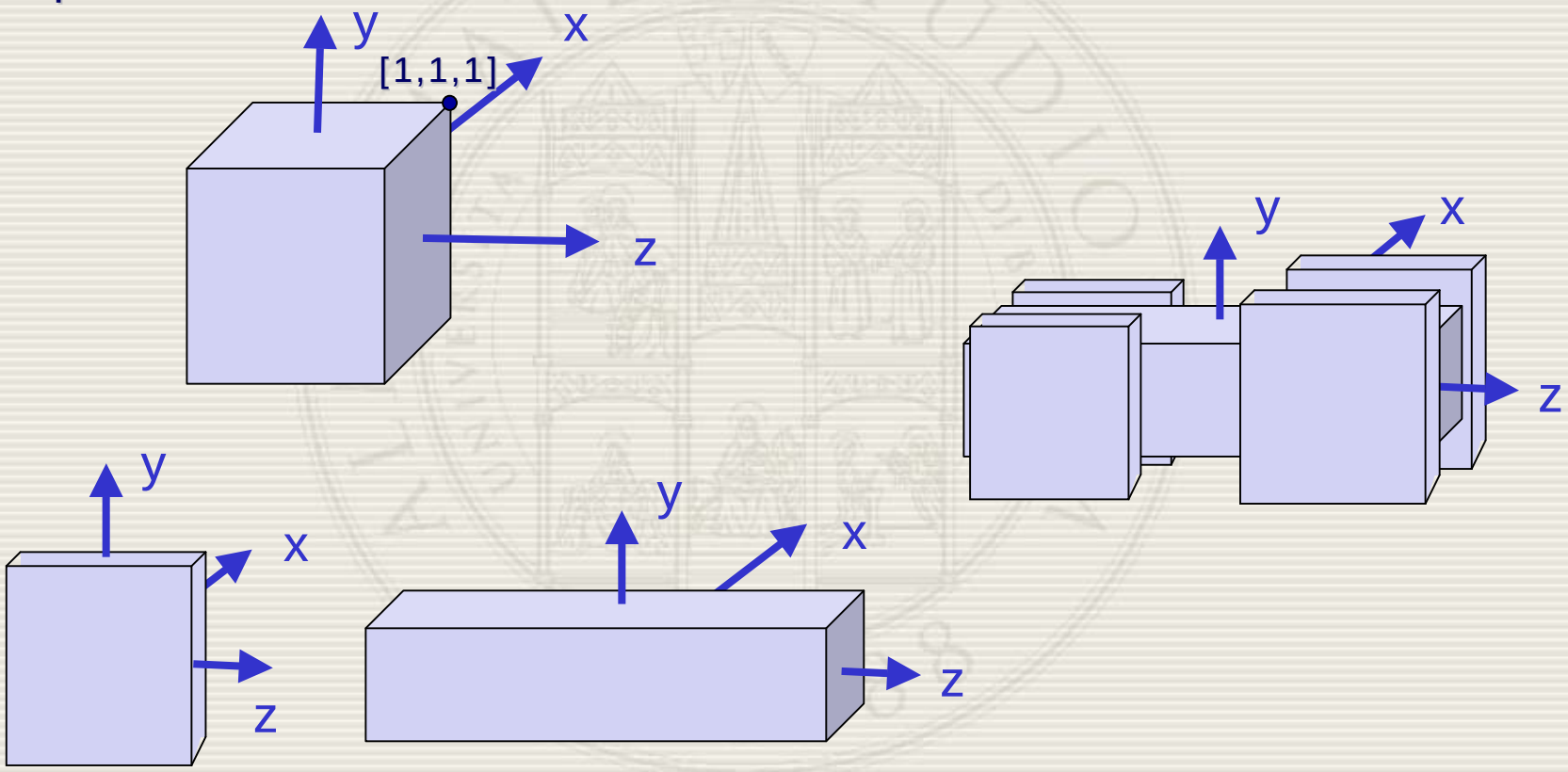
In questa fase non gestiamo il tempo.

```
int done=0;
  while (!done) {

    // guardo se c'e' un evento:
    if (SDL_PollEvent(&e)) {
     // se si: processa evento
     switch (e.type) {

      ......

case SDL_MOUSEMOTION:
       if (e.motion.state & SDL_BUTTON(1) ) {
         viewAlpha+=e.motion.xrel;
         viewBeta +=e.motion.yrel;
         if (viewBeta<-90) viewBeta=-90;
         if (viewBeta>+90) viewBeta=+90;
         // richiedi ridisegno
         redraw();
       }
       break;
     }
} else {
     // richiedi ridisegno
     redraw();
   }
  }
```

Codice cartella progettoCar1

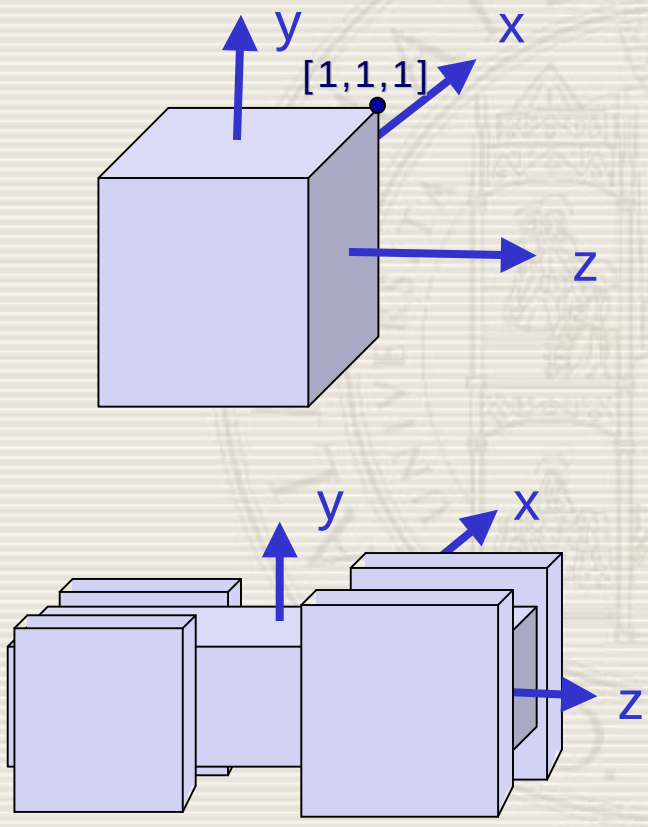
Passo 1

Usiamo come semplice forma per l'auto (carrozzeria e 4 ruote) 5 parallelepipedi; inizializziamo una sola geometria cubo che scaleremo e trasleremo nello spazio per dar forma alle varie parti.



Passo 1

Costruiamo una function drawCube() che definisce un cubo di centro l'origine e vertci [±1, ±1, ±1]; quindi trasliamo, scaliamo e disegnamo le varie parti

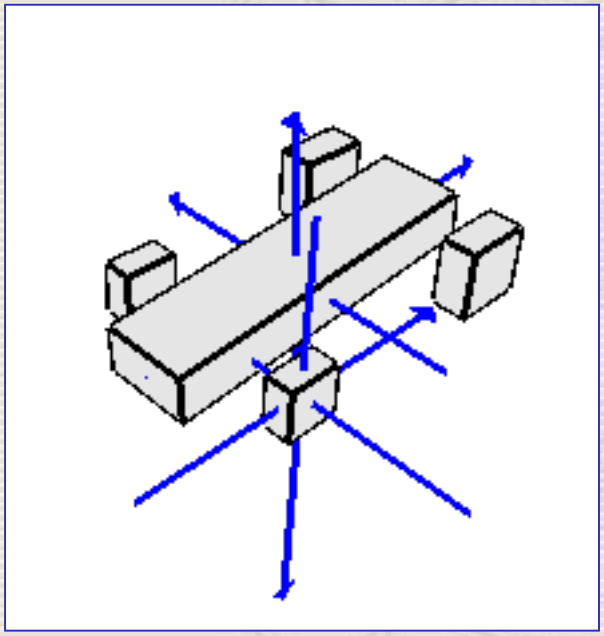



```
// vado al frame pezzo_A
  glScalef(0.25 , 0.14 , 1);
  drawCube();

// ruota posteriore D
  glTranslatef( 0.58,-0.05,+0.8);
  glScalef(0.1, 0.20, 0.20);
  drawCube();

// ruota posteriore S
  glTranslatef(-0.58,-0.05,+0.8);
  glScalef(0.1, 0.20, 0.20);
  drawCube();

// ruota anteriore D
  glTranslatef( 0.58,-0.05,-0.55);
  glScalef(0.08, 0.15, 0.15);
  drawCube();

// ruota anteriore S
  glTranslatef(-0.58,-0.05,-0.55);
  glScalef(0.08, 0.15, 0.15);
  drawCube();
```

Passo 1

Concentriamoci sulle ruote; vogliamo farle ruotare.

```
//inizializziamo mozzo
float mozzo=0
```



```
} else {
    // nessun evento: siamo IDLE
    mozzo--;
    // richiedi ridisegno
    redraw();
  }
}
```

```
// vado al frame pezzo_A
  glScalef(0.25 , 0.14 , 1);
  drawCube();

// ruota posteriore D
  glTranslatef( 0.58,-0.05,+0.8);
  glRotatef(mozzo,1,0,0);
  glScalef(0.1, 0.20, 0.20);
  drawCube();

// ruota posteriore S
  glTranslatef(-0.58,-0.05,+0.8);
  glRotatef(mozzo,1,0,0);
  glScalef(0.1, 0.20, 0.20);
  drawCube();

 // ruota anteriore D
  glTranslatef( 0.58,-0.05,-0.55);
  glRotatef(mozzo,1,0,0);
  glScalef(0.08, 0.15, 0.15);
  drawCube();

 // ruota anteriore S
  glTranslatef(-0.58,-0.05,-0.55);
  glRotatef(mozzo,1,0,0);
  glScalef(0.08, 0.15, 0.15);
  drawCube();
```

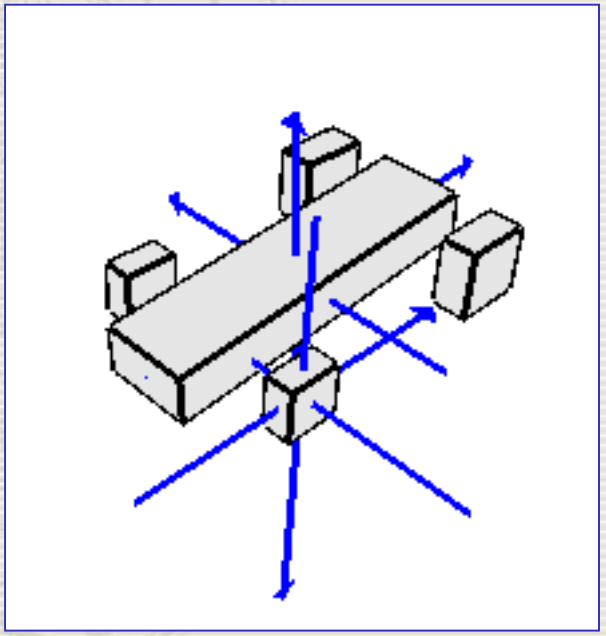
Passo 2

Usando la stessa 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 C++ così da poter definire una classe Car e una classe Controller.

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

Aggiungiamo ZoomIn e ZoomOut con rotella del mouse.



Codice cartella progettoCar2

Passo 2: la fisica

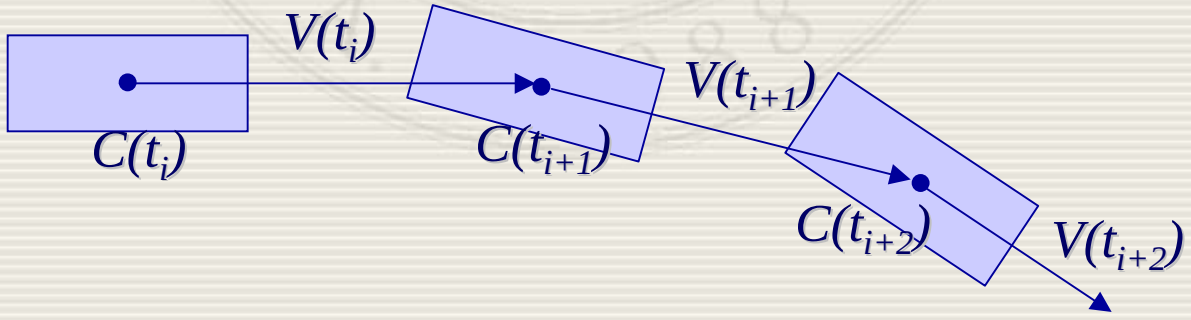
Faremo uso dell'equazione del moto uniformemente accelerato

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

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

$$C(i+1) = C(i) + V(i) \text{ con } V(i)=V(i-1)+a$$

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.

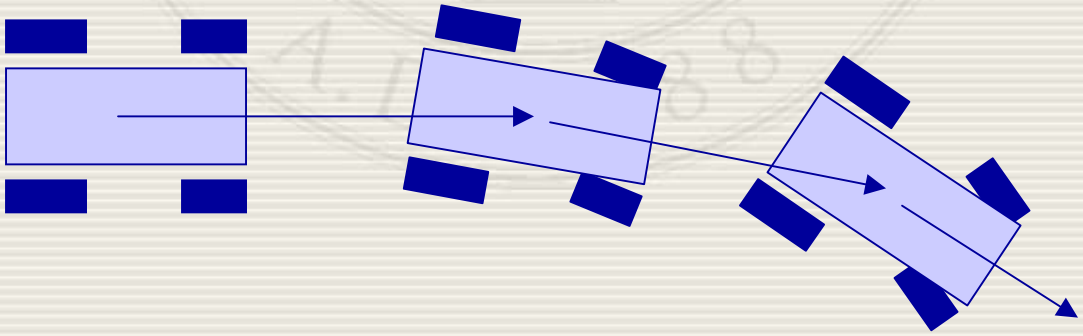


Passo 2: la fisica

Determinata la posizione dell'auto $C(i+1)$, per l'istante successivo $i+1$ dovremo applicare le opportune trasformazioni geometriche per posizionare la geometria correttamente 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 .

```
float sterzo=0;

  velSterzo=3.4;            // A
  velRitornoSterzo=0.93; // B, sterzo massimo = A*B/(1-B) = 45.17 gradi

  //gestione dello sterzo
    sterzo=sterzo ± velSterzo; //+ sterzo a sinistra – a destra
    sterzo*=velRitornoSterzo; // ritorno a volante fermo; (...(A*B+A)*B+..+A)*B
```



Passo 2: la fisica

$$\lim_{n\to\infty}(\ldots(AB+A)B+A)B+\ldots+A)B=$$

$$\lim_{n\to\infty}(AB+AB^2+AB^3+\ldots+AB^n)=$$

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

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

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

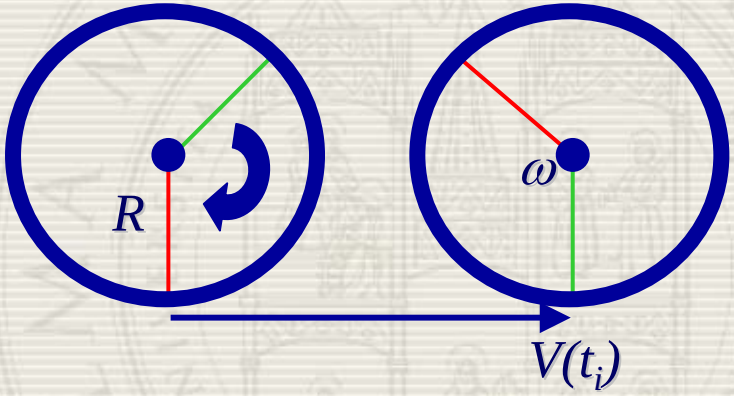
$$\lim_{n\to\infty}\frac{AB}{(1-B)}(1-B^n)=\frac{AB}{(1-B)} \qquad \lim_{n\to\infty}B^n=0 \;\; se \;\; 0\langle B\langle 1$$

Passo 2: la fisica

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

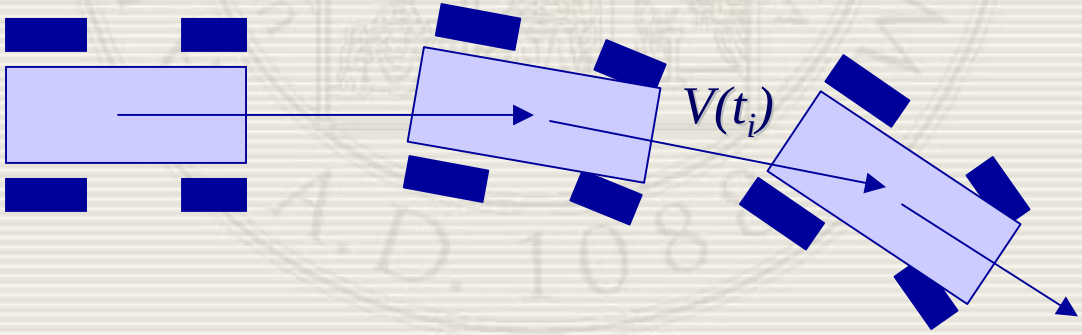
$$\omega = V/R$$

con R raggio della ruota.



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

Angolo in gradi

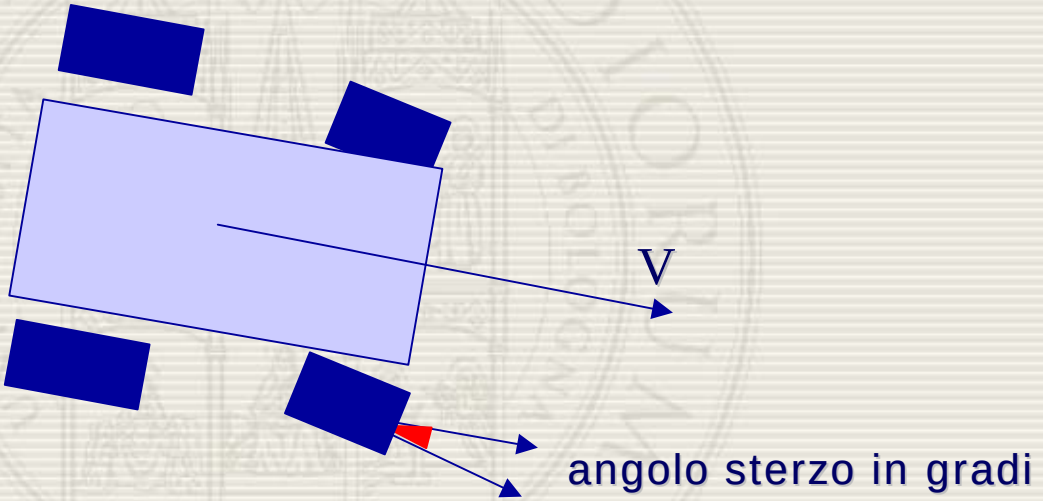


L'angolo ω (in radianti) deve essere trasformato in gradi: $\omega * 180/\pi$

Passo 2: la fisica

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

facing = ($|V|$*grip)*(angolo sterzo)



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

Passo 2: la fisica

```
void Car::Init(){
  // 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
  controller.Init();

  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 nullo

  // 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
}
```

classe Car
metodo Init

Passo 2: la fisica

```
// DoStep: facciamo un passo di fisica (a delta-t costante)
void Car::DoStep(){
  // computiamo l'evolversi della macchina

  float vxm, vym, vzm; // velocita' in spazio macchina

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

  // gestione dello sterzo
  if (controller.key[Controller::LEFT]) sterzo+=velSterzo;
  if (controller.key[Controller::RIGHT]) sterzo-=velSterzo;
  sterzo*=velRitornoSterzo; // ritorno a volante fermo

  if (controller.key[Controller::ACC]) vzm-=accMax;  // accelerazione in avanti
  if (controller.key[Controller::DEC]) vzm+=accMax; // accelerazione indietro

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

classe Car
metodo DoStep

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)
float 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;
}
```

classe Car
metodo DoStep

Passo 2: la fisica

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

```
int nstep=0; // numero di passi di FISICA fatti fin'ora
const int PHYS_SAMPLING_STEP=10; //numero di millisec
                                 //che un passo di fisica simula
```

```
} else {
    // nessun evento: siamo IDLE

    Uint32 timeNow=SDL_GetTicks(); // che ore sono?
    bool doneSomething=false;
    int guardia=0; // sicurezza da loop infinito

    // finche' il tempo simulato e' rimasto indietro rispetto al tempo reale...
    while (nstep*PHYS_SAMPLING_STEP < timeNow ) {
      car.DoStep();
      nstep++;
      doneSomething=true;
      timeNow=SDL_GetTicks();
      if (guardia++>1000) {done=true; break;} // siamo troppo lenti!
    }

    if (doneSomething) {
      redraw();
    }
    else { // tempo libero!!!
    }
  }
}
```

main loop

Passo 2: la fisica

Gestiamo lo ZoomIn e lo ZoomOut con la rotella del mouse.

```
case SDL_MOUSEWHEEL:
        if (e.wheel.y < 0 ) {
          // avvicino il punto di vista (zoom in)
          eyeDist=eyeDist*0.9;
          if (eyeDist<1) eyeDist = 1;
        };
        if (e.wheel.y > 0 ) {
          // allontano il punto di vista (zoom out)
          eyeDist=eyeDist/0.9;
         };
        break;
```

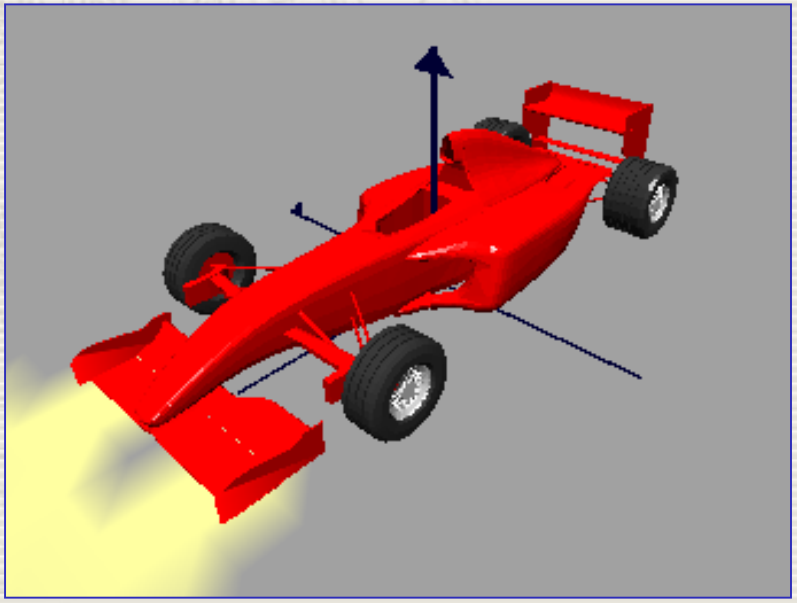
main loop

Passo 3

Viene usata una classe 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.



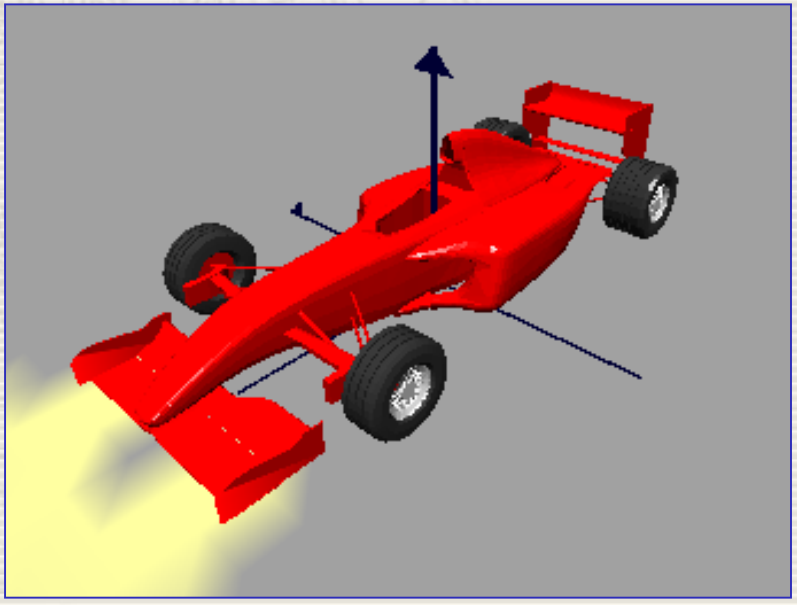
Codice cartella progettoCar3

Passo 3

Si introduce un piano (stradale) (già nella versione progettoCar2 era presente un piano) e si vogliono aggiungere e gestire due fari anteriori (questo comporta la ridefinizione del piano stradale).

Si vuole gestire l'input anche con un joypad (velocità e sterzo graduali).

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

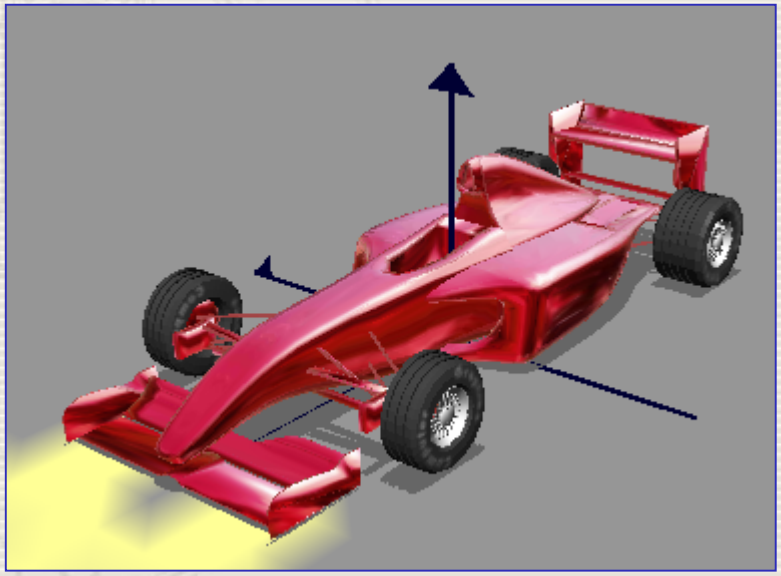


Passo 4

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

Si vogliono introdurre delle texture:
1. environment mapping dell'auto;
2. projection di una scritta sulle gomme;
3. environment mapping del cielo (texture cielo per creare una scena realistica ambientata all'aperto).

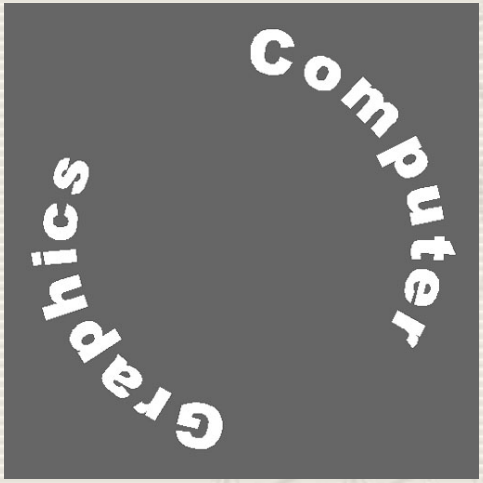
Si introducono le ombre ed in particolare l'ombra dell'auto e delle ruote sul pavimento.



Codice progettoCar4

Passo 4

Le texture usate:

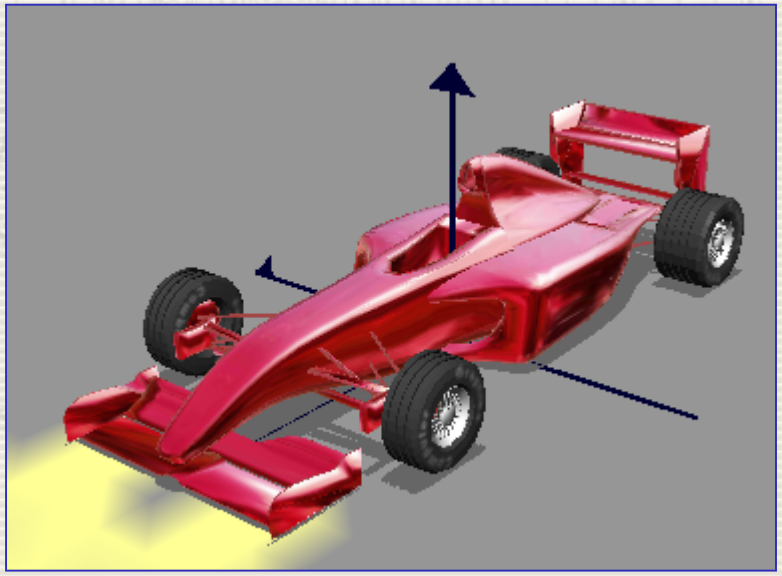


Passo 4

Gestione di più Camere che permettano di seguire l'auto da diverse posizioni fra cui quella del pilota.

Visualizzazione anche in wireframe e possibilità di abilitare/disabilitare:

1. differenti camere
2. tipi di rendering
3. texturing/colore
4. fari
5. ombre



Passo 4

Si aggiunga una pista (circuito chiuso)

Si preveda una fase demo in cui l'auto viene guidata da un pilota automatico (parametri predefiniti di controllo dell'auto)