# Gran Turismo Simulator

DSD :: GAMES

# ANALISI GTS: Gran Turismo Simulator

**Indice:**

# 1) PRESENTAZIONE

## 1.1 DESCRIZIONE E SCOPO DEL GIOCO

Gts è un gioco simulativo di corse d'auto. Il giocatore veste i panni di un pilota professionista del campionato Gran Turismo. Il giocatore avrà a disposizione un parco auto dal quale scegliere il mezzo e una serie di circuiti. Le auto sono rigorosamente di serie, non ci sono auto elaborate o modificate specificamente. Inoltre, sono tutte dotate dei sistemi elettronici di controllo della frenata (ABS) e della stabilità (ESP e ASR). Selezionata l'auto e il circuito dove gareggiare, il giocatore può iniziare la corsa.
Il fine del gioco è quello di realizzare il miglior tempo per completare un giro di ogni pista. Ogni pista, mantiene memorizzato il record (il tempo più basso). Non ci sono avversari diretti; ovvero durante la corsa, non ci sono altre auto nel circuito. L'unico nemico da combattere è il tempo, l'unico avversario è il record da superare. Segue che è prioritaria una guida pulita e priva di sbavature. La filosofia del gioco è di premiare una guida corretta, penalizzando ogni uscita fuori pista. Per rendere più realistico il gioco non ci sono auto con cambi automatici (che selezionano il rapporto senza alcun comando del pilota).
La simulazione, per semplicità, implica delle clausole.

## 1.2 CLAUSOLE

- Non ci sono auto con cambio automatico.
- Tutte le auto dispongono di ABS, ESP ed ASR (o equivalenti).
- Tutte le auto dispongono di limitatori ed è impossibile rompere il motore.
- L'auto deve essere riportata intatta dopo la corsa.

## 1.3 CARATTERISTICHE

- Genere: Simulatore d'auto 3D
- Piattaforma: PC
- Requisiti minimi: processore a 800 MHz, ram di 64Mbyte, scheda video da 32Mbyte.
- Motore 3D: Dsd Games QTT (Quads To Triangle)

## 1.4 SPECIFICHE TECNICHE

Il gioco è stato programmato in linguaggio C++ in ambiente Microsoft Visual C++ 6.0 del Microsoft Visual Studio. La grafica 3D è stata implementata tramite le librerie grafiche OpenGL.

# 2) STRUTTURA DEL GIOCO

## 2.1 GESTIONE SIMULAZIONE FISICA

La simulazione tende a essere quanto più realistica possibile. La necessità di avere auto che dispongono dei sistemi di controllo elettronici della frenata e della stabilità, è dovuta al fatto che la simulazione non gestisce la dinamica di un'auto in moto non controllato; in pratica non gestisce il sovrasterzo, il sottosterzo, le sbandate, i testacoda, le sgommate, i danni da urto, ed altri fenomeni non controllabili dal pilota. Con i dispositivi elettronici di controllo, limitiamo lo stile di guida e semplifichiamo notevolmente la simulazione, senza rinunciare alla realisticità. Come scritto sopra, non gestiamo in alcun modo i danni da urto e soprattutto i rimbalzi contro i muri. Per ovviare, inseriamo la clausola che l'auto deve essere riportata intatta nel garage dopo la corsa. Questo significa che non appena l'auto tocca un muro, anche si striscio, il gioco si riavvia. E ciò, ancora una volta, per penalizzare uno stile di guida scorretto e soprattutto guadagnare di realisticità, infatti è luogo comune, in molti giochi simili, che un'auto si scontri frontalmente contro un muro ad elevata velocità, e subito dopo riprenda la corsa come se nulla fosse accaduto.
Altro aspetto fondamentale è l'assenza del comando della frizione. Il gioco simula che ogni volta che cambiamo marcia, abbiamo usato implicitamente la frizione, e soprattutto nel modo corretto.

## 2.2 PARAMETRI DEL GIOCO:

La simulazione tiene conto di moltissimi parametri.

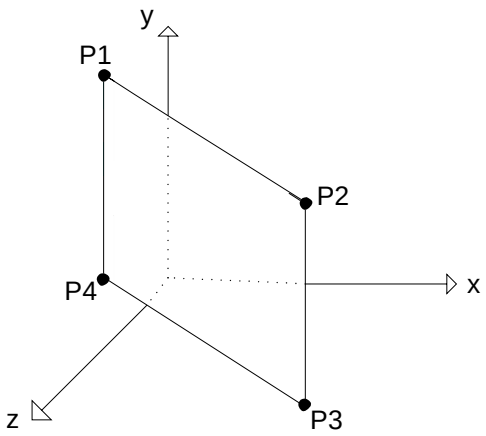
- *Tempo di cambiata [ms]:* tempo necessario per cambiare marcia
- *Velocità massima in 1° marcia [Km/h]:* velocità massima consentita in 1° marcia, prima che intervenga il limitatore di giri.
- *Velocità massima in 2° marcia [Km/h]:* velocità massima consentita in 2° marcia, prima che intervenga il limitatore di giri.
- *Velocità massima in retromarcia [Km/h]:* velocità massima consentita in retromarcia, prima che intervenga il limitatore di giri.
- *Freno motore [Km/h]:* velocità massima che è consentito frenare (scalando semplicemente di marcia, senza usare il comando freno) con il freno motore senza superare il limite di giri massimo del motore.
  Esempio:
  Marcia: 4°, giri/min: 6000, max giri: 7000. Se scalo di marcia, sicuramente il numero di giri supererà il massimo, e romperei il motore. Non è consentito rompere il motore.
- *Minimo [giri/min]:* indica il regime minimo del motore
- *Velocità di risposta dello sterzo [x/ms]:* velocità con la quale lo sterzo ruota da un lato all'opposto.
- *Ritardo di risposta [ms]:* ritardo di tempo da quando attiviamo il comando sterzo a quando effettivamente lo sterzo ruota.
- *Attrito [%]:* coefficiente che va da 0 a 100 ad indicare l'attrito dell'asfalto sulle gomme. Se è uguale a 0 abbiamo un sistema inerziale, se è uguale a 100 l'auto non si muove neanche accelerando.

## 2.3 L'AMBIENTE 3D

### 2.3.1 I POLIGONI

La pista è l'ambiente tridimensionale che il motore grafico disegna. La pista è formata da una serie di poligoni. Per uniformità, i poligoni avranno sempre 4 vertici. Questo non significa necessariamente che nel mondo che creiamo, ci saranno solo rettangoli o quadrati; possiamo creare un triangolo infatti, ripetendo due volte lo stesso vertice.
I vertici, dato che l'ambiente è 3D, hanno 3 coordinate (X,Y,Z).

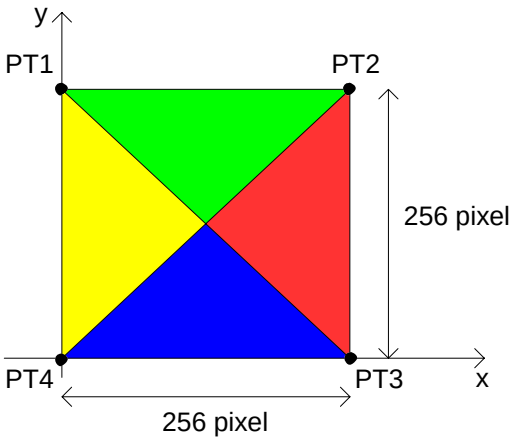
Pn ( Xn , Yn , Zn )




Una volta creato un poligono, esso deve essere colorato per essere visibile. Qui intervengono le texture.

### 2.3.2 LE TEXTURE

Le texture sono delle immagini bitmap, di dimensioni ben precise, che vengono usate come "sfondo" del poligono. Il poligono infatti, ha una sua superficie; questa viene riempita con una immagine. L'immagine bitmap usata come texture, deve essere quadrata, cioè deve avere lo stesso numero di pixel per ogni lato. Inoltre, un altro vincolo fondamentale, è che il numero di pixel di risoluzione sia una potenza del 2. Pertanto, noi useremo immagini bitmap da 256*256 pixel.



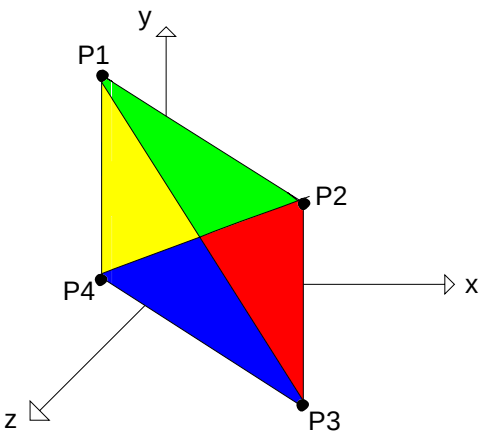

Un'immagine, per definizione, è a 2 sole dimensioni.

PTn ( Xn , Yn )

Per integrarle in un poligono a 3 dimensioni, occorre associare, per ogni vertice del poligono, un vertice della texture.

Pn ( Xn, Yn, Zn ) → PTn ( Xn, Yn )

In questo modo, avremo un poligono 3D con una superficie colorata in 2D.




Con questa tecnica, siamo in grado di costruire un mondo tridimensionale di poligoni. Con più poligoni, possiamo costruire di tutto (piramidi, cubi, parallelepipedi, prismi etc). L'unica cosa che non possiamo creare sono superfici perfettamente rotonde o curve.
Come già detto, tutta la pista è creata da poligoni. Le informazioni dei poligoni di una pista, sono memorizzate in un file specifico per ogni pista, chiamato World.txt. Ovviamente, useremo più texture. Esse sono nominate con un numero crescente che va da 0 a un certo numero B. Esempio (0.bmp ; 1.bmp ; 2.bmp ; N.bmp)

### 2.3.3 WORLD.TXT : un mondo di poligoni

Il file è di tipo testuale. Esso è strutturato in due zone, zona Parametri e zona Poligoni. La zona parametri è situata sulla prima linea di testo del file, ed ha la seguente struttura:
"PARAMETRES: A B"
A è un numero intero che indica il numero di poligoni della pista.
B è un numero intero che indica il numero di texture che utilizza la pista.
La zona Poligoni è immediatamente successiva e contiene i vari poligoni. Ogni poligono ha la seguente struttura:
//Nome
C
PTx PTy Px Py Pz
PTx PTy Px Py Pz
PTx PTy Px Py Pz
PTx PTy Px Py Pz

Nome identifica il poligono. E' preceduto da una coppia di /, ad indicare che è un commento. Infatti, la procedura di caricamento del file, quando trova la coppia di /, salta il rigo e passa al successivo. Il nome può essere anche omesso.
C è un numero intero che indica il numero della texture che associamo al poligono. (Esempio, se c'è il numero 3, associamo al poligono la texture nominata 3.bmp)
PTx e PTy indicano le coordinate 2D del vertice della texture che associamo al vertice del poligono
Px, Py e Pz sono le coordinate 3D del vertice del poligono.
Come già detto, i poligoni hanno 4 vertici, pertanto ci saranno 4 righe.
Con questa semplice struttura, abbiamo definito un mondo 3D.

Tutti i dati relativi alla pista, sono localizzati all'interno della cartella PistaN, con N che indica il numero della pista, e sono:

- Le immagini bitmap usate come texture
- Il file world.txt
- Il file structure.txt
- Il file record.txt

### 2.3.4 STRUCURE.TXT

Il file structure è un file di testo che è adibito a memorizzare le seguenti informazioni sulla pista:

- Indica il tipo di suolo
- Indica la localizzazione dei muri

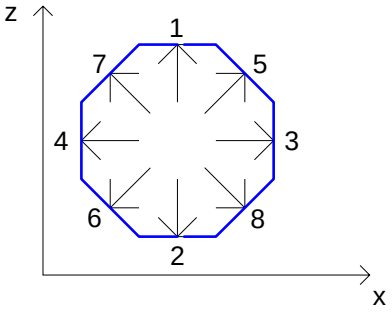
La pista, ha due tipi di suolo.

- L'asfalto è il suolo dove l'auto si muove regolarmente. La sua superficie è piatta e con pendenza nulla.
- Il prato (o simili) è il suolo dove l'auto normalmente non deve correre, dato che l'aderenza, la frenata, la stabilità, ed altri parametri ne risentono negativamente. Inoltre la superficie è irregolare e causa vibrazioni.

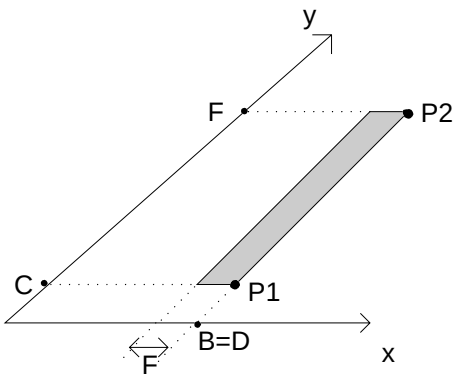
L'auto, pertanto, o è posizionata sull' asfalto o sul prato. Non esistono altri tipi di suolo.
La pista è delimitata da muri, l'auto non può attraversare e neanche toccare un muro (vedi le clausole).
Analizziamo la struttura di una singola riga di informazioni:
//Nome
A B C D E F
Nome identifica la riga di informazioni. E' sotto commento e non viene usato nella simulazione.
A è un numero intero e indica il tipo di informazione della riga
B,C,D,E,F sono numeri naturali (float) che assumono diversi significati in base al tipo di informazione

#### 2.3.4.1 I MURI

I muri verranno identificati come una regione della superficie della pista di forma rettangolare. Quando l'auto, entra in questa regione, vuol dire che sta attraversando un muro.
La gestione dei muri raccoglie i numeri da 1 a 8 del tipo di informazioni della riga.



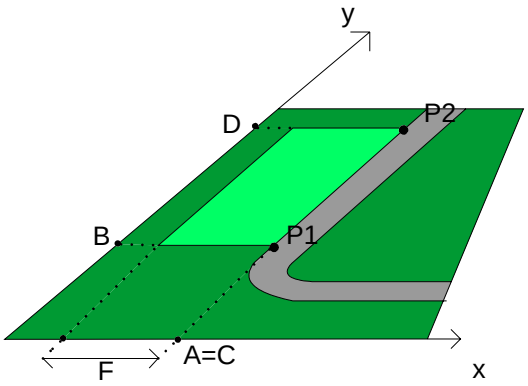

La figura sopra mostra un ottagono di colore blu. Ogni suo lato rappresenta un tipo di muro. A ogni lato è associato un numero e una freccia. Il lato num1, ad esempio, indica che stiamo per dare informazioni su un muro che è parallelo all'asse x e che è attivato quando l'auto lo attraversa con z crescente (da sotto verso sopra).
I muri infatti, per come sono gestiti, funzionano come dei filtri. Permettono il passaggio in un verso, ma non nell'altro. Ovviamente, noi li gestiremo in modo tale che in nessun verso sia possibile attraversarli. Lo stesso ragionamento va fatto per tutti gli altri lati dell'ottagono.
Specifichiamo che i lati obliqui indicano un muro obliquo con una angolazione qualsiasi (non necessariamente di 45° come nell'ottagono, ma un angolo compreso tra 1° e 89°).
Quando il parametro A, assume un valore compreso tra 1 e 8, i parametri B,C,D,E,F assumono i seguenti valori:



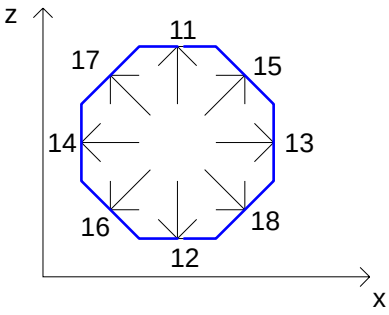

B e C indicano rispettivamente i valori di x e z del punto P1
D e F indicano rispettivamente i valori di x e z del punto P2
F indica la profondità del muro
Durante la fase del ciclo del sistema dei controlli, se la posizione dell'auto risulta interna all'area specificata, viene attivato un flag che riavvia la simulazione.

### 2.3.4.2 I TIPI DI FONDO

Se la gestione dei muri è risultata semplice, quella dei tipi di asfalto non è lo è stata affatto. Per non creare un altro file, abbiamo dovuto usare la stessa struttura della gestione dei muri. Praticamente, inseriamo dei muri "virtuali", dato che nel mondo 3D non ci sono, che delimitano i confini dell'asfalto con il prato. E' come se una zona rettangolare di prato, fosse un muro.

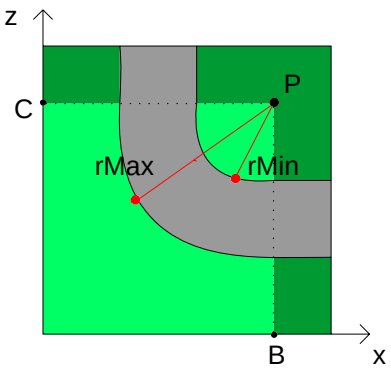



L'area di color grigio è la pista di asfalto. Intorno c'è il prato verde. Supponiamo di definire l'area color verde chiaro come una zona di prato, come un muro di vertici P1 e P2 e profondità F. Con questa nuova gestione, dobbiamo anche cambiare il numeri del tipo di informazioni, che diventano:

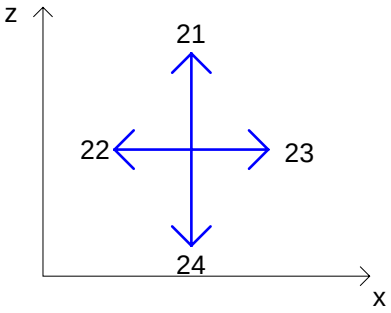



Se pertanto il parametro A ha un valore compreso tra 11 e 18, indica che stiamo definendo un muro virtuale per la gestione dei tipi di asfalto. Quando la procedura di controlli rileva che l'auto è entrata in un muro virtuale, si attiva un flag che indica che l'auto non è più sull'asfalto, ma è andata fuori pista. Tutto questo va bene se la pista non ha curve. Infatti, i muri sono in ogni caso dei rettangoli e non possono essere curvi.

Per gestire i tipi di fondi durante una curva, inseriamo una nuova serie di tipi di informazioni.




L'area di color grigio indica la pista di asfalto. L'area di color verde indica il prato. Per delineare in modo preciso i confini tra asfalto e pista nella curva , indichiamo un punto P e due raggi. rMin indica il raggio minimo, esso delinea il confine interno della curva. rMax indica il raggio massimo e delinea il confine esterno della curva. Durante i controlli, quando l'auto entra in una curva, viene calcolata la sua distanza dal punto P. Se questa distanza, diventa minore di rMin o maggiore di rMax, l'auto ha superato i confini della curva e viene attivato il flag di fuori pista.
I parametri B,C,D,E,F assumono i seguenti significati:
B e C indicano rispettivamente i valori di x e z del punto P.
D indica rMin
E indica rMax
F è inutilizzato
Ovviamente, differenziamo il tipo di informazione in base alla direzione della curva.




Quindi, se il tipo di informazione A ha un valore compreso tra 21 e 24, esso identifica informazioni per gestire i tipi di fondo durante una curva.

## 2.4 L'AUTO

La cartella auto contiene i file relativi al parco auto.
Ogni auto ha i seguenti parametri:
Foto
Nome
Potenza [CV]
Coppia motrice [Nm]
Velocità massima [Km/h]
Numero di giri massimo [giri/min]
Numero di marce (escluso la retromarcia)
Tenuta [%]
Frenata [%]
Peso [Kg]
Rapporti del cambio.

Analizziamo i parametri Tenuta e Frenata. Dato che nella realtà i parametri che caratterizzano un sistema frenante in un'auto sono molti (Frenata da 100 Km/h, da 150 Km/h, affaticamento, corsa del pedale etc…), noi riassumiamo il tutto in un solo coefficiente espresso in % (da 0 a 100), ad indicare 0% un impianto frenante inesistente, a 100% il miglior impianto frenante fino ad ora disponibile in commercio per auto di serie. Stesso ragionamento vale per la tenuta, che indica la tenuta di strada, importante per non uscire fuori pista durante una curva.
Altra caratteristica importante è la rapportatura del cambio, ovvero la simulazione tiene conto per ogni marcia, del rapporto che il cambio ha. Il rapporto è espresso in X:1, ad indicare quanti giri X del motore, occorrono per completarne uno sull'asse in uscita dal cambio, diretto poi al differenziale e al riduttore. Ovviamente più è alto il numero X, più la marcia è bassa e viceversa.

Ogni auto presenta i seguenti file:

- CarN.bmp : foto esterna dell'auto
- ParN.bmp : immagine che descrive le caratteristiche dell'auto
- CarN.txt : file contenente i parametri dell'auto

Le immagini CarN.bmp e ParN.bmp sono utilizzate per fornire la descrizione dell'auto che si sceglie. Nel gioco infatti, si potrà cambiare in qualsiasi istante l'auto con cui gareggiare.
Le auto sono numerate in modo crescente da 0 a N. Il numero di auto del parco auto è indicato nel file Auto.txt.

### 2.4.1 AUTO.TXT

Il file auto.txt è un file di testo con una sola riga, con la seguente struttura:
"NUM_AUTO: n"
dove n indica il numero di auto da caricare nel motore grafico.
Ogni qualvolta si aggiunge una nuova auto nel parco auto, si deve aggiornare questo parametoi che, altrimenti, impedirebbe di usare la nuova auto.

### 2.4.2 CARN.TXT

Il file carN.txt è un file di testo con la seguente struttura:
"Name: stringa_nome"
"Potenza: num"
"Coppia: num"
"Max_Speed: num"
"Max_Giri: num"
"Max_Marcie: num"
"Tenuta: num"
"Frenata: num"
"Peso: num"
"R: num"

“1: num”
“2: num”
“3: num”
“4: num”
“5: num”
“6: num”

stringa_nome indica il nome dell’auto che verrà visualizzata durante la simulazione. Num indica un generico numero float. Tutti i parametri sono di facile comprensione. I rapporti delle marce sono indicati con dei numeri crescenti da 1 a 6 (R per la retromarcia). Si deve inserire il numero di giri che compie l’albero motore per ottenere un giro dell’asse in uscita dal cambio. Vediamo che il numero massimo di marce consentito è 6 (escluso la retromarcia), se l’auto ha meno di 6 marce, si deve inserire il valore 0.0 nel rapporto non consentito.

## 2.5 LE STRUTTURE DEI DATI

Tutti le informazioni che servono al gioco, devono essere caricate in memoria centrale (RAM). Pertanto, tutte le immagini usate come texture e tutti i file di testo, devono essere localizzati e memorizzati in apposite strutture. Delle procedure apposite implementeranno questa fase. Analizziamo le strutture dei dati:

### 2.5.1 STRUTTURA DELL’AUTO: tagAUTO

```
typedef struct tagAUTO
{
        char name[40];
        float xpos, zpos;
        float dir;
        float speed;
        float acc;
        float max_speed;
        float peso;
        float brake;
        float tenuta;
        float potenza;
        float coppia;
        float giri;
        float max_giri;
        float marce[9];
        int marcia;
        int max_marcie;

} AUTO;
```

La struttura auto, crea un buffer di memoria, che viene aggiornato costantemente in base ai parametri dell’auto.

- *xpos e zpos,* indicano le coordinate dell’auto sul piano x-z. La simulazione, infatti, non prevede salite o discese, e l’auto si muoverà solamente nelle 2 dimensioni.
- *dir* indica la direzione che ha l’auto in ° nel singolo instante
- *speed* indica la velocità in m/s nel singolo instante
- *acc* indica l’accelerazione dell’auto nel singolo instante
- *max_speed* indica la velocità massima che l’auto può raggiungere
- *peso* indica il peso dell’auto in kg
- *brake* indica il coefficiente di frenata
- *tenuta* indica il coefficiente di tenuta
- *potenza* indica il potenza dell’auto in kw
- *coppia* indica la coppia dell’auto in nm (newtometri)

- *giri* indica il numero di giri del motore nel singolo instante
- *max_giri* indica il numero massimo di giri che sopporta il motore
- *marce[7]* è un array che contiene la rapportatura del cambio per ogni marcia (l'indice 0 è per la retromarcia, 1 per la 1° marcia, 2 per la 2° marcia e così via)
- *marcia* indica la marcia del cambio dell'auto nel singolo instante
- *max_marcie* indica il numero massimo di marce del cambio

Prima di avviare il ciclo della simulazione, vengono caricati in memoria tutti i dati relativi al parco auto. Pertanto creeremo un array di buffer AUTO chiamato cars[]. Ogni buffer dell'array, contiene le informazioni relative a un'auto. Ad esempio, il buffer cars[3] contiene le informazioni lette dal file car3.txt.
La procedura che apre i file carN.txt e memorizza i dati in cars[] è: SetupAuto()

### 2.5.2 STRUTTURA DI UN VERTICE: tagVERTEX

Vertex definisce un buffer di memoria che contiene le informazioni relative a un generico vertice di un poligono con una texture.

```
typedef struct tagVERTEX
{
        float x, y, z;
        float u, v;
} VERTEX;
```

x,y,z indicano i valori delle 3 dimensioni
u,v indicano i valori delle 2 dimensioni dei vertici della texture

### 2.5.3 STRUTTURA DI UN MURO: TagSTRUCTURE

```
typedef struct tagSTRUCTURE
{
        int type;
        float d;
        float xa, ya, xb, yb;
} STRUCTURE;
```

type indica il tipo di informazione associati alle altre variabili.
d,xa,ya,xb,yb indicano rispettivamente i parametri B,C,D,E,F descritti in precedenza.

### 2.5.4 STRUTTURA DI UN POLIGONO: TagTRIANGLE

Triangle definisce un buffer di memoria che contiene le informazioni relative a un generico poligono (non necessariamente un triangolo, è solo una specifica del motore grafico QTD).

```
typedef struct tagTRIANGLE
{
        VERTEX vertex[4];
        int textu;
} TRIANGLE;
```

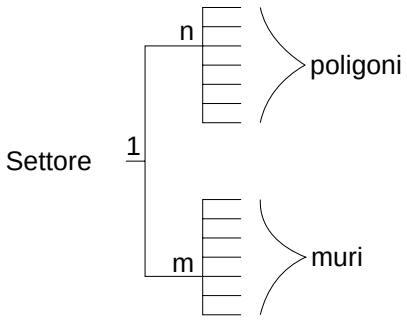
nell'array di buffer vertex[] conteniamo le informazioni relativi ai 4 vertici del poligono
textu indica il numero della texture associata al poligono

### 2.5.5 STRUTTURA DELLA PISTA: tagSECTOR

Il settore indica il mondo che andiamo a creare con i poligoni. Noi associamo a un settore, una pista.

```
typedef struct tagSECTOR
{
        int numtriangles;
        int numstru;
        TRIANGLE* triangle;
        STRUCTURE* stru;
} SECTOR;
```

numtriangles indica quanti poligoni ci sono nella pista
numstru indica quanti muri (compresi quelli virtuali) ci sono nella pista
*TRIANGLE* triangle;* definisce un puntatore a una lista di buffer TRIANGLE
*STRUCTURE* stru;* definisce un puntatore a una lista di buffer STRUCTURE




## 2.6 CARICAMENTO DEI DATI

Tutte le strutture sino ad ora analizzate servono in quanto ci saranno delle apposite procedure, che si occuperanno di reperire i dati da appositi file e di allocarli in memoria. Le funzioni sono:

- SetupWorld(): per il caricamento dei dati della pista dal file world.txt
- SetupAuto(): per il caricamento dei dati delle auto dai file carN.txt

Entrambe le funzioni hanno la stessa struttura. Prima leggono i parametri all'inizio del file e successivamente, tramite un ciclo, leggono i dati e li allocano nelle strutture.

### 2.6.1 CARICAMENTO DELLE TEXTURES

Anche le texture devono essere caricate in memoria di massa. Le librerie OpenGL definiscono un tipo di buffer adatto alla memorizzazione di immagini bitmap con risoluzione di 256*256 pixel. Noi definiremo 5 categorie di buffer per le texture:

```
GLuint texture[3][20];
GLuint car_textu[20];
GLuint car_textu2[20];
GLuint font_textu[2];
GLuint game_textu[6];
```

texture[n][m]
la matrice di buffer texture[n][m] memorizza le immagini relative alla pista. L'indice m indica la texture caricata, ad esempio texture[n][3] contiene i dati relativi all'immagine 3.bmp della pista. L'indice n indica il tipo di filtro utilizzato nel caricare la texture. I filtri da noi usati sono 3 e vi rimandiamo allo specifico paragrafo. Notiamo da subito un limite, il numero massimo di texture caricabile è 20 per una pista 20. Un provvedimento che se da un lato limita la fantasia dei programmatori, dall'altra velocizza la simulazione.
car_textu[n]
è un array di buffer che memorizza i dati relativi alle foto delle auto (carN.bmp).
car_textu2[n]

è un array di buffer che memorizza i dati relativi alle immagini descrittive dell'auto (parN.bmp)
game_textu[n]
è un array di buffer che memorizza i dati relativi alle immagini (loghi e credits) del gioco.
font_textu[n]
è un array di buffer che memorizza i dati relativi all'immagine del font.

Le procedure che memorizzano i dati delle immagini nei buffer sono:
LoadGLMapTextures() : carica le texture della pista
LoadGLCar1Textures() : carica le texture relative alle foto dell'auto
LoadGLCar2Textures() : carica le texture relative alle immagini descrittive dell'auto
LoadGLFontTextures() : carica le texture relative ai font
LoadGLGameTextures() : carica le texture relative al gioco

Le procedure hanno tutte la stessa impostazione. Leggono dal rispettivo file quante e quali texture caricare, avviano un ciclo nel quale le memorizzano negli appositi buffer e se tutto è andato bene, ritorna un valore positivo.

## 2.7 I FILTRI QUALITA'

Le OpenGL implementano l'uso dei filtri. I filtri vengono applicati alle texture e ne modificano alcune caratteristiche. Nel gioco usiamo 3 filtri, che gestiscono la qualità. Analizziamoli:

- Filtro bassa-qualità
  Il filtro bassa-qualità usa il filtro denominato Nearest. Come suggerisce il nome, esso lavora in funziona della distanza dalla telecamera della texture. Più è maggiore la distanza, più la texture perde di dettaglio e qualità. Essa viene compressa con un fattore che varia in modo proporzionale alla distanza. Ovviamente, quando saremo molto vicini alla texture con la telecamera, essa ci apparirà perfetta. Man mano che ci allontaniamo, in modo graduale e impercettibile, la texture si "sfoca". Se si applica questo filtro, il Motore grafico è molto veloce a disegnare la scena. Se da un lato, infatti, si usa del tempo per la compressione delle texture, dall'altro, esso si recupera abbondantemente, grazie al fatto che il motore grafico, successivamente, dovrà proiettare in ambiente 3D un numero considerevolmente minore di dettagli.
- Filtro media-qualità
  Il filtro media-qualità è associato al filtro denominato MipMapped. Il filtro non apporta modifiche alla texture, esso infatti crea una mappa dei bit, del tutto identica a quella di una immagine bitmap. Questo vuol dire che l'immagine non è compressa. In fase di caricamento, risulta veloce. In fase di proiezione, il Motore grafico ha un numero considerevole di bit da disegnare e risulta più lento.
- Filtro alta-qualità
  Il filtro alta-qualità è associato al filtro denominato Linear. E' il filtro più "pesante" da gestire per una scheda grafica. Esso infatti, oltre a non comprimere le texture, le migliora in base a una serie di parametri. In parole semplici, esso linearizza la scena, la rende più omogenea e gratificante alla vista. Smussa gli spigoli vivi, rende le immagini più "soft", con una leggerissima sfocatura per eliminare gli errori di proiezione. Ad esempio, se ci avviciniamo molto a una texture, con il filtro media qualità, notiamo chiaramente i pixel che la compongono; con il filtro alta-qualità, questo non avverrà mai. Tra i parametri fondamentali ci sono la distanza della telecamere dalla texture, l'angolo di visuale e l'illuminazione (che nel gioco non è implementata).

La necessità di adoperare i filtri nasce dal fatto che i PC non sono tutti uguali in termini di prestazioni. Il gioco infatti deve poter funzionare correttamente nelle più svariate configurazioni. Su PC poco potenti useremo il filtro bassa-qualità e viceversa.

## 2.8 CICLO DI SISTEMA

Il ciclo di sistema è il ciclo generale della simulazione. Completando il passaggio di un ciclo, il gioco compie le seguenti operazioni (in ordine):

1. Controlla se è stato usato qualche comando
2. Calcola la posizione dell'auto
3. Disegna la scena

Il ciclo viene comandato da una variabile booleana chiamata "done". Se questa è positiva, il ciclo viene interrotto.

### 2.8.1 CONTROLLO COMANDI

Per controllare se l'utente ha usato un comando del gioco, usiamo il seguente codice:

```
if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))
	{
		if (msg.message==WM_QUIT)
		{
			done=TRUE;
		}
		else
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}
```

La funzione PeekMessage(&msg,NULL,0,0,PM_REMOVE) ci restituisce un valore positivo, se ha ricevuto un qualsiasi messaggio di input dalla tastiera. Il messaggio ricevuto viene memorizzato nella struttura msg. Se il messaggio ricevuto è un messaggio di uscita, la variabile done viene settata positiva. Altrimenti, il messaggio viene codificato e inviato alla struttura msg. Tramite questa conversione, possiamo controllare lo stato di ogni tasto tramite un array booleano di 256 unità, chiamato "keys[n]". La conversione infatti attiva la cella corrispondente al tasto premuto. Se keys['A'] è attivo è stato premuto il tasto 'A'. Da qui iniziamo una serie di controlli per ogni tasto, a cui abbiamo associato un particolare compito.

### 2.8.2 CALCOLA LA POSIZIONE DELL'AUTO

Questa parte del ciclo, si divide in due sezioni. La prima, è una serie di condizioni e formule che calcolano l'esatta posizione dell'auto in base a vari parametri.
La seconda è una vera e propria procedura, che controlla se l'auto ha toccato un muro e il tipo di fondo della pista.

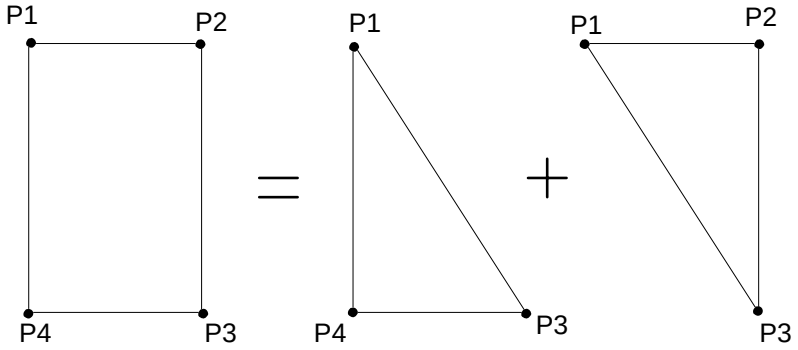
*Prima sezione.*
Comando GAS
Premendo il tasto FrecciaSu, viene aumentato il numero di giri del motore secondo la formula:
giri = giri + potenza * coppia * coefficiente_marcia / (peso *8)
In fase di accelerazione, viene aumentato il numero di giri. Successivamente viene calcolata la velocità in base al numero di giri.
Comando Frena
Premendo il tasto FrecciaGiù, viene ridotta la velocità dell'auto secondo la formula:
velocità = velocità – coefficiente_frenata/(velocità*kost)
Con questa formula, più andiamo veloci, più è difficile rallentare, in modo inversamente proporzionale. Diminuita la velocità, viene calcolato il numero di giri.
Una volta ricavati velocità e numero di giri, si possono calcolare le coordinate dell'auto nello spazio:
x = x – cos(direzione) * velocità
y = y – sen(direzione) * velocità

Direzione, ovviamente, viene gestita dai comandi dello sterzo.
Le formule che abbiamo visto sono molto semplificate rispetto a quelle reali del gioco. Nel gioco infatti teniamo conto di altri numerosissimi vincoli, come il numero di giri massimo e il numero di giri minimo, l'accelerazione massima per non sgommare etc. A questo scopo c'è il limitatore.

*Seconda sezione.*
Una volta calcolate le coordinate dell'auto, si deve verificare che essa non stia violando qualche regola, in particolare che stia correndo su asfalto e che non stia toccando un muro. A questo scopo c'è la funzione CheckPosition() che esegue i controlli e in base al loro risultato attiva o disattiva dei flag (come fuori_pista).

### 2.8.3 DISEGNA LA SCENA
La procedura dell'Engine grafico che disegna la scena è la: DrawGLScene().
Essa tramite un ciclo legge dalle apposite strutture i poligoni e le texture e li disegna sullo schermo. E' in questa procedura che si implementa la tecnica QTT dell'Engine.

## 2.9 ENGINE GRAFICO 3D QTT BY Dsd::Games
Il motore grafico è una procedura che disegna il mondo. E' di tipo QTT (Quads to Triangle), una specifica della Dsd Games. In pratica, esso converte poligoni a 4 vertici in poligoni a 3 vertici. In particolare converte un poligono a 4 vertici, in 2 poligoni a 3 vertici. Questo perché il processore della scheda grafica, impiega molto meno tempo a disegnare un poligono a 3 vertici, di uno a 4.

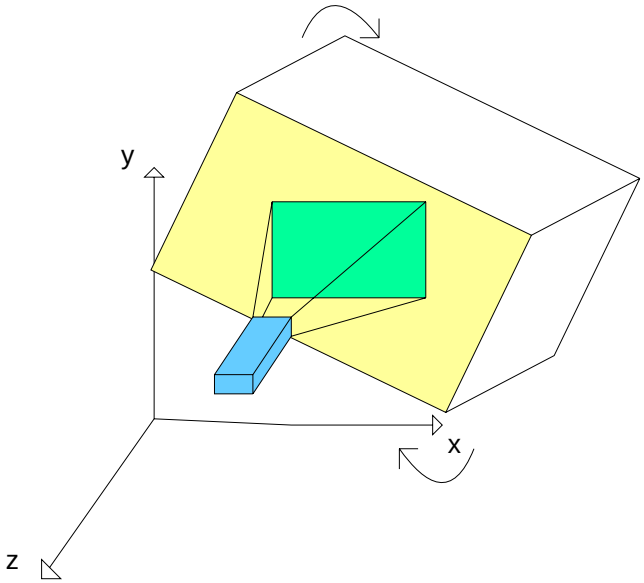



Per questo motivo, si chiama QTT, Quadrati convertiti in Triangoli.

## 2.10 GESTIONE DEL MOVIMENTO
Le OpenGL sono una raccolta di librerie per la gestione della grafica 2D e 3D. Nell'ambiente 3D, esse permettono di definire un "mondo" fatto di poligoni e di osservarlo tramite una "videocamera". Ci sono diversi punti da definire. Le OpenGL lavorano su coordinate assolute. Quando definiamo i vertici dei poligoni, essi devono essere assoluti rispetto all'origine degli assi. Il "mondo assoluto" che noi costruiamo, non lo vedremo mai nello schermo, ma è un concetto immaginario. (E' come se volessimo vedere nella realtà tutte le facce di un cubo!)
Una volta definito il "mondo assoluto", possiamo muoverci con la videocamera per osservarlo in prospettiva e da qualsiasi angolazione. Per creare il movimento dell'auto abbiamo due scelte:

- Muoviamo la telecamera in base ai movimenti dell'auto e il mondo resta fisso
- Muoviamo il mondo in base ai movimenti dell'auto e la telecamera resta fissa

Entrambe le tecniche portano allo stesso risultato. Noi adotteremo la 2° che è più semplice. Nei videogiochi dell'ultima generazione, invece, si usa la 1° strada, in quanto sorge la necessità di avere più videocamere.
Per prima cosa, stabilizziamo la videocamera in un punto fisso. Dopodiché, iniziamo a traslare e ruotare il mondo in base ai comandi usati dall'utente.
In particolare, per girare a destra o sinistra, noi ruoteremo il mondo in direzione opposta, intorno all'asse y della videocamera, (se premiamo frecciaSINISTRA, ruotiamo il mondo verso destra). Per andare avanti o indietro con l'auto, trasliamo sull'asse x della videocamera il mondo in direzione opposta (se premiamo frecciaSU, trasliamo il mondo indietro).




In questo modo, non è l'auto a muoversi, ma è il mondo a muoversi in direzioni opposte.
Per effettuare le traslazioni, useremo la funzione:
glTranslatef(delta_x, delata_y, delta_z)
questa funzione trasla il mondo sui rispettivi assi delle componenti delta_x,y e z.
Se vogliamo traslare di 20 unità l'asse x: glTranslatef(20, 0, 0)
Se vogliamo traslare di 20 unità l'asse x e di 5 l'asse z: glTranslatef(20, 0, 5)
Come possiamo notare, si possono effettuare traslazioni multiple.

Per effettuare le rotazioni, useremo la funzione:
glRotatef(angolo,x,y,z)
questa funzione ruota il mondo sugli assi specificati nei parametri. Angolo indica di quanti angoli si vuole ruotare il mondo. x,y e z indicano l'entità della rotazione sui 3 assi dimensionali.
Se si vuole ruotare di 90° l'asse y : glRotatef(90,0,1,0)
Se vogliamo ruotare di 45° l'asse x e z : glRotatef(45,1,0,1)
Se vogliamo ruotare di 45° l'asse x, di 90° l'asse y e di 135° l'asse z : glRotatef(45,1,2,3)
Come possiamo notare, si possono effettuare ruotazioni multiple.

Per effettuare le proiezioni, c'è la fase di rendering grafico. In questa fase le OpenGL costruiscono delle matrici di vertici che trasformano attraverso una serie di procedimenti matematici. Queste matrici vengono allocate in appositi buffer e successivamente serviranno per stampare su video i poligoni. Per gestire i buffer, useremo la tecnica del double buffering.

## 2.11 IL DOUBLE BUFFERING

Il double buffering è una tecnica molto usata nei giochi di grafica 3D. Si creano 2 buffer identici, uno adibito a disegnare i poligoni sullo schermo reale, e uno su uno schermo virtuale che non vediamo. Il concetto è che mentre si esegue il rendering della scena successiva, si visualizza il buffer della scena precedente, ma si lavora sul buffer virtuale. Quando il rendering è completo, si swappano i buffer; si invertono, e in un instante, si passa dalla scena vecchia a quella nuova. In questo modo le operazioni di disegno non vengono visualizzate. La nuova scena appare in modo istantaneo e la simulazione guadagna fluidità.
La procedura usata è:
SwapBuffers(hDC)
hDC è il nome del buffer virtuale che si definisce nell'intestazione del programma

## 2.12 IL LIMITATORE

Il limitatore è uno strumento che adoperiamo per limitare i controlli dell'utente, per evitare situazioni indesiderate, quali lo spegnimento dell'auto, la fusione del motore, una sgommata etc. Quando esso interviene, sullo schermo apparirà una scritta di avviso. Esso interviene anche nella gestione dello sterzo. Per impedire bruschi cambi di traiettoria ad alte velocità, lo sterzo cambia le sue caratteristiche in base alla velocità. In particolare cambia l'angolo di sterzata massimo, la velocità di sterzata e il ritardo di risposta. Ad alte velocità, l'angolo di sterzata diventa minimo, la velocità di sterzo viene ridotta e viene aumentato il ritardo di risposta. In questo modo, ad alte velocità, lo sterzo è totalmente controllabile. Se lo sterzo non prevedesse questa utile opzione, ad alta velocità, il minimo tocco del comando, ci farebbe girare di molto, perdendo il controllo.
I limiti dell'angolo di sterzata, vengono visualizzati nella parte inferiore dello schermo.

## 2.13 I COMANDI

Comandi generali

- H : Visualizza l'Help
- C : Cancella record della pista
- I : attiva/disattiva parametri tecnici
- M : visualizza info Mappa

Comandi auto

- FrecciaSU : Comando gas
- FrecciaGIU : Comando freno
- FrecciaDESTRA: Comando sterzo destra
- FrecciaSINISTRA: Comando sterzo sinistra
- A : Comando cambio marcia superiore
- Z : Comando cambio marcia inferiore
- Q : Seleziona auto precedente
- W : Seleziona auto successiva

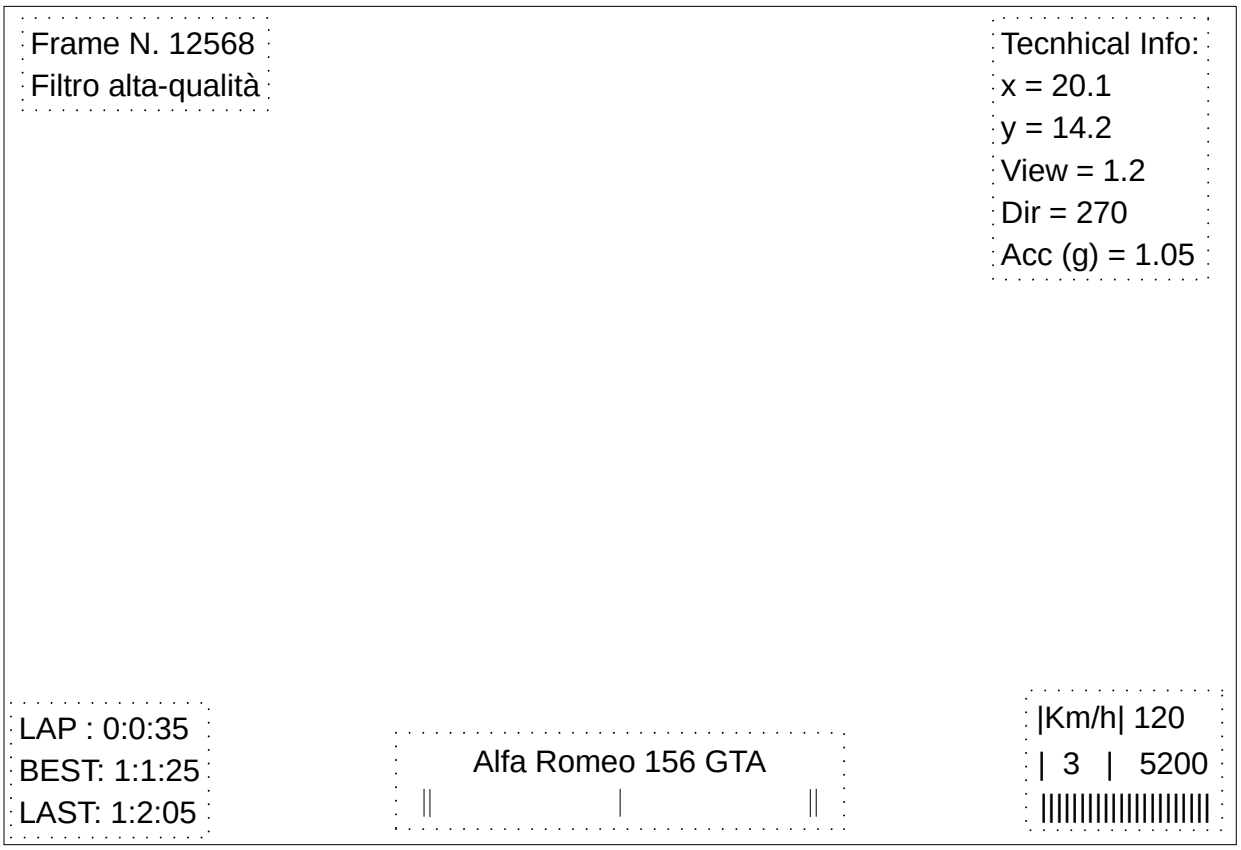
Comandi Engine3D

- F1 : attiva/disattiva FullScreen
- F : seleziona filtro qualità
- N : attiva/disattiva modo NoClip
- S* : aumenta altezza da terra
- X* : diminuisci altezza da terra
- PageUp* : aumenta angolo di visuale
- PageDown* : diminuisci angolo di visuale

I tasti segnati * sono disponibili solo in modo NoClip

## 2.14 LA SCHERMATA DI GIOCO

Quando il caricamento del gioco è completato, la schermata si presenta in questo stato:




Sulla parte inferiore dello schermo, abbiamo i classici indicatori.
Sulla destra ci sono velocità in Km/h, marcia inserita e numero di giri.
Sulla sinistra ci sono i tempi: Lap indica il giro che si sta completando (aggiornato ogni frame); best indica il record e last indica il tempo dell'ultimo giro eseguito. (al primo giro, Last non c'è)
Al centro, viene visualizzato il nome dell'auto che si sta usando e la visualizzazione della posizione dello sterzo e dei limiti di tenuta imposti dal limitatore.
Sulla parte superiore dello schermo, abbiamo dei parametri nascosti, che si possono visualizzare usando il tasto I.
Sulla sinistra, abbiamo il numero di frame che abbiamo fino ad ora disegnato sullo schermo e il tipo di filtro in uso.
Sulla destra, abbiamo le coordinate della posizione dell'auto (X e Y), angolo di visuale (View), la direzione dell'auto (Dir) e l'accelerazione istantanea in G.

### 2.15 GESTIONE DEI RECORD

Il fine ultimo del gioco è quello di eseguire con la propria auto un giro record. Ogni pista ha il suo file record.txt che contiene il record in millisecondi. Durante il gioco, viene misurato il tempo per eseguire ogni giro. A tal proposito, ci sono degli accorgimenti per evitare che un pilota tagli la pista. In diversi punti di ogni pista, ci sono dei flag che vengono attivati quando l'auto ha superato una certa zona, Se, ad esempio, c'è una galleria che all'interno ha un flag, e il pilota la taglia per abbreviare la strada, il flag non si attiva e il giro viene annullato.
RECORD.TXT
Il file record è un file testuale con la seguente struttura:
"RECORD: n"
dove n indica il numero di ms (milli-secondi) impiegati per completare un giro della pista. Ogni volta che si supera il record, il file viene aggiornato.

## 3) CONCLUSIONI

Con questo concludiamo l'analisi di questo gioco simulativo, il primo gioco 3D della Dsd::Games, che fino ad ora aveva creato solo giochi 2D (in ordine cronologico):

- VideoPoker
- Snake
- Itis Attack
- Biliardo Simulator

## 4) RINGRAZIAMENTI

Ringraziamento a NeHe per aver creato i tutorial di programmazione con OpenGL.
Ringraziamento a Etrusco Muratore per aver tradotto i tutorial di NeHe in italiano.
Ringraziamento a www.gameprog.it per aver messo a disposizione i tutorial di NeHe