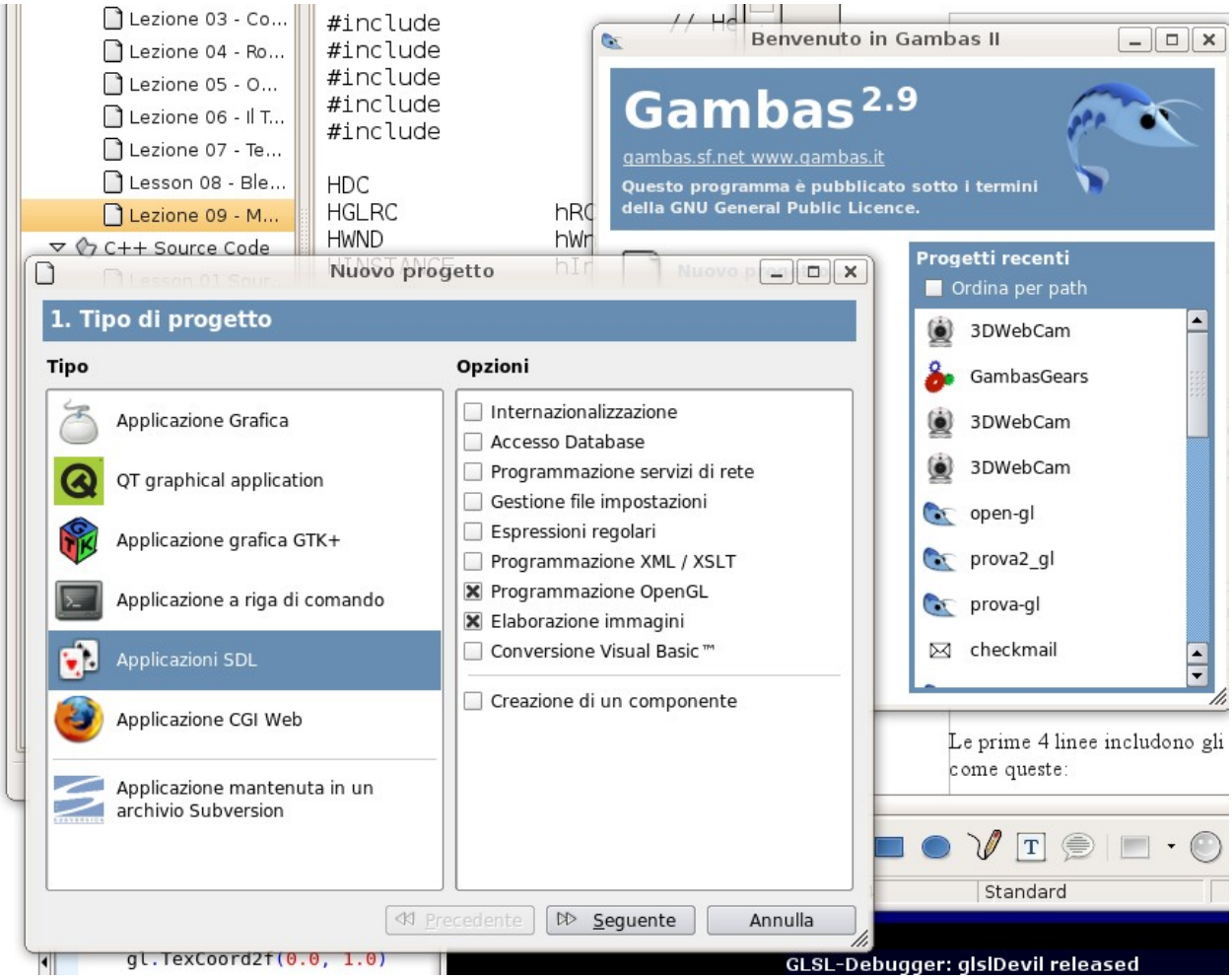
Gambas e opengl

tratto dalla guida di Jeff Molofee ed adattato a gambas da Fea Sergio f.surfing@tiscali.it

Benvenuti in questo minitutorial sulle OpenGL.in gambas.
Io non sono un programmatore ma un semplice appassionato di gambas, cercando informazioni su opengl e gambas su internet, mi sono imbattuto in una semplice guida scritta da Jeff Molofee (http://nehe.gamedev.net) sull' uso delle opengl con il classico C ++, ho cercato quindi di modificare il codice per l' uso con il nostro amato Gamberetto e riscriverlo in questa guida che segue la struttura e i testi originali.

Lezione 1

Comincerò questo tutorial saltando subito al codice.
La prima cosa che dobbiamo fare è costruire un progetto Gambas di tipo SDL con selezionate le opzioni opengl ed elaborazioni immagini



Dopo aver dato il nome all' applicazione e scelto la cartella in cui salvarlo ci troveremo con un modulo Mmain in cui andremo a scrivere il nostro codice.

Le prime 3 linee includono le dichiarazioni delle variabili e costanti globali utilizzate , nel nostro caso avremo la costante ScrWidth che indica la larghezza che avrà la nostra finestra, mentre ScrHeight ne indica l' altezza , poi andremo a dichiarare e setatre la variabile Screen di tipo Window

```
  ' Gambas module file

PRIVATE CONST ScrWidth AS Integer = 640
PRIVATE CONST ScrHeight AS Integer = 480
PRIVATE Screen AS NEW Window(TRUE) AS "Screen"

'La prossima sezione di codice definisce la funzione Main() ovvero l' inizio del nostro programma,

PUBLIC SUB Main()

'Ora andremo a definire la dimensione dello schermo

:
  screen.Width = ScrWidth
  screen.Height = ScrHeight

'E lo visualizziamo:

  Screen.show()

'definiamo il tipo di bordo della finestra

Screen.Border = Screen.Resizable

'ora andremo a richiamare la funzione per inizializzare la finestra opengl

InitGL()

e chiudiamo la funzione corrente

END

'la prossima sezione sarà appunto quella di inizializzazione che abbiamo chiamato initGL:


PUBLIC SUB InitGL()

'La prossima linea abilita lo smooth shading. Lo smooth shading sfuma i colori lungo i lati di un
'poligono e perfeziona l'illuminazione. Spiegherò lo smooth shading in maggior dettaglio in un altro
'tutorial.
 Gl.ShadeModel(gl.GL_SMOOTH)

'Le seguenti linee stabiliscono il colore dello schermo quando si cancella. Se non sapete come
'funzionano i colori ve lo spiegherò rapidamente. La scala di valori per il colore va da 0.0 a 1.0.
' 0.0 indica il più scuro e 1.0  indica il più chiaro. Il primo parametro dopo gl.ClearColor è
'l'intensità del rosso, il secondo parametro è per il verde e il terzo per il blu. Più il valore si avvicina
' a 1.0, più sarà luminoso il colore specificato. L'ultimo numero indica un valore Alpha. Quando si
```

```
'tratta di cancellare lo schermo, possiamo non preoccuparci sul quarto valore. Per adesso lasciatelo a
' 1.0. Spiegherò il suo utilizzo in un altro tutorial.
'Si creano colori differenti mescolando i tre colori primari (rosso, verde, blu). Spero lo abbiate
'imparato a scuola. Quindi, se abbiamo glClearColor(0.0 , 0.0 , 1.0 , 0.0) cancelleremo lo schermo
' con un colore blu luminoso. Se abbiamo glClearColor(0.5  , 0.0 , 0.0 , 0.0) cancelleremo lo
' schermo con un colore rosso medio. Non luminoso (1.0) e non scuro (0.0). Per fare uno sfondo
' bianco, dovrete impostare tutti i valori al massimo possibile (1.0). Per fare uno sfondo nero dovrete
' impostare tutti i colori al minimo possibile (0.0).

Gl.ClearColor(0.0, 0.0, 0.0, 1.0)
Gl.Clear(gl.GL_COLOR_BUFFER_BIT AND gl.GL_DEPTH_BUFFER_BIT)


'Le tre linee seguenti hanno a che fare con il Depth Buffer. Immaginate il depth buffer come strati
' (layers) nello schermo. Il depth buffer tiene conto di quanto gli oggetti siano in profondità nello
' schermo. Non useremo veramente il depth buffer in questo programma, ma quasi ogni programma
' OpenGL che disegna sullo schermo in 3D utilizza il depth buffer. Esso seleziona quale oggetto
' disegnare per primo in modo che un quadrato che avete disegnato dietro ad un cerchio non vi
' finisca sopra. Il depth buffer è una parte molto importante delle OpenGL.

Gl.ClearDepth(1.0)
Gl.Enable(gl.GL_DEPTH_TEST)
Gl.DepthFunc(gl.GL_LEQUAL)

'Ora diremo alle OpenGL che vogliamo che venga eseguita la migliore correzione prospettica.
' Questo causa un piccolissimo calo delle prestazioni, ma rende la visuale prospettica migliore a
' vedersi.

Gl.Hint(gl.GL_PERSPECTIVE_CORRECTION_HINT, gl.GL_NICEST)

'a questo punto chiudiamo la funzione

END


'Il compito della prossima sezione di codice è di ridimensionare la scena OpenGL ogni volta che la
' finestra (dando per scontato che state utilizzando una finestra piuttosto che la modalità fullscreen)
' viene ridimensionata. Persino se non c'è la possibilità di ridimensionare la finestra (per esempio, si
' è in modalità fullscreen), questa routine verrà chiamata almeno una volta quando il programma
' parte per settare la visuale prospettica. La scena OpenGL verrà ridimensionata basandosi sulla
' larghezza e l'altezza della finestra in cui viene visualizzata.
'Definiamo due variabili ,   ratio che indica il rapporto tra larghezza e l' altezza della finestra  e
'Height che indica l' altezza della stessa

DIM ratio AS Float
DIM Height AS Integer

  Height = Screen.Height

  IF Height = 0 THEN Height = 1

  ratio = Screen.Width / Height
```

```
Gl.Viewport(0, 0, Screen.Width, Screen.Height)
'Le linee seguenti impostano lo schermo per una visuale prospettica. Ciò significa che le cose più
' sono distanti più diventano piccole. Questo crea una scena dall'aspetto realistico. La prospettiva
' viene calcolata con un angolo di osservazione di 45° basato sull'altezza e la larghezza della
' finestra. 0.1, 100.0 sono il punto di partenza e di arrivo di quanto in profondità è possibile
' disegnare nello schermo.

        Gl.MatrixMode(gl.GL_PROJECTION)
        Gl.LoadIdentity()
        Glu.Perspective(45.0, ratio, 0.1, 100.0)

'glMatrixMode(GL_PROJECTION) indica che le 2 linee successive del codice influiranno sulla
' matrice di proiezione. La matrice prospettica si occupa di aggiungere la prospettiva alla nostra
' scena. glLoadIdentity() è simile ad un reset. Ripristina lo stato di partenza nella matrice
' selezionata. Dopo che glLoadIdentity() è stata chiamata, staabiliamo la visuale prospettica per la
' scena. glMatrixMode(GL_MODELVIEW) indica che ogni nuova trasformazione influirà sulla
' matrice di visualizzazione del modello. La matrice di visualizzazione del modello è dove vengono
' memorizzate le informazioni riguardanti il nostro oggetto. Infine azzeriamo la matrice di
' visualizzazione del modello. Non preoccupatevi se non capite questa parte, la rispiegherò in un
' altro tutorial. Sappiate solo che questo DEVE essere fatto per una buona scena prospettica.


        Gl.MatrixMode(gl.GL_MODELVIEW)
        GL.LoadIdentity()
'ed infine chiudiamo la funzionerà

END


' In questa sezione andrà tutto il codice relativo al disegno. Ogni cosa avete in mente di mostrare a
' schermo andrà in questa sezione di codice. Tutti i tutorial dopo di questo aggiungeranno codice in
' questa sezione del programma. Se già sapete qualcosa di OpenGL, potete provare a creare forme
' basilari aggiungendo del codice OpenGL dopo glLoadIdentity() e prima di WAIT. Se siete
' nuovi di OpenGL, aspettate i miei prossimi tutorial. Per ora tutto quello che faremo sarà cancellare
' lo schermo col colore che abbiamo deciso in precedenza, cancellare il depth buffer e azzerare la
' scena. Non disegneremo ancora niente.

PUBLIC SUB Screen_draw()
        gl.ClearColor(0, 0, 0, 1)
        Gl.Clear(gl.GL_COLOR_BUFFER_BIT AND gl.GL_DEPTH_BUFFER_BIT)
        Gl.LoadIdentity()
        WAIT 0.01
END

'Nell asezione seguente andremo ad assegnare dei compiti a alla pressione di alcuni tasti, per ora
' utilizzeremo il tasto F1 per passare dalla finestra al fullscreen ed il tasto ESC per uscire
' dall'applicazione
PUBLIC SUB Screen_keyPress()

  IF key.Code = key.F1 THEN Screen.Fullscreen = NOT Screen.Fullscreen
  IF key.Code = key.Esc THEN Screen.Close()
```

END

Lezione 2

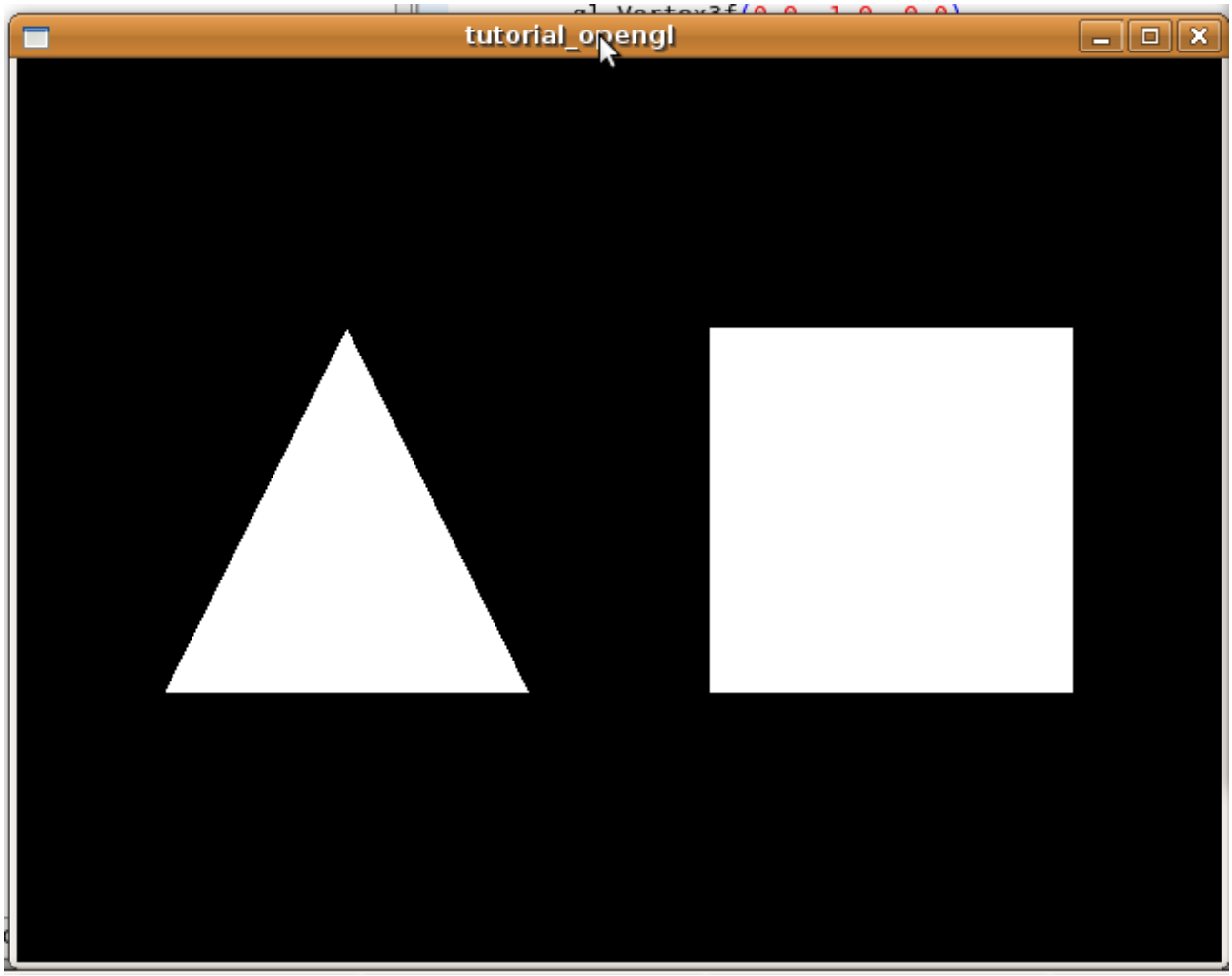
```
' Nel primo tutorial abbiamo imparato come creare una finestra OpenGL. In questo tutorial
'impareremo a creare sia triangoli che quadrati. Creeremo un triangolo utilizzando
'GL_TRIANGLES e un quadrato utilizzando GL_QUADS.

'Utilizzando il codice del primo tutorial, ne aggiungeremo alla procedura   Screen_draw()
' Riscriverò l'intera preocedura qui sotto. Se volete modificare l'ultima lezione, potete sostituire la
'procedura  Screen_draw()  col codice qui sotto, o semplicemente aggiungere le linee di codice che
' non ci sono.

PUBLIC SUB Screen_draw()


        gl.ClearColor(0, 0, 0, 1)
        Gl.Clear(gl.GL_COLOR_BUFFER_BIT AND gl.GL_DEPTH_BUFFER_BIT)

        Gl.LoadIdentity()

'Quando chiamate una glLoadIdentity() quello che state facendo è muovervi al centro dello schermo
' con l'asse X che va da sinistra a destra, l'asse Y dall'alto verso il basso, e l'asse Z che va dentro e
' fuori lo schermo. Il centro di uno schermo OpenGL è 0.0f sugli assi X e Y.
'Alla sinistra del centro abbiamo numeri negativi. Verso destra avremo numeri positivi.
' Muovendoci verso l'alto avremo un numero positivo, verso il basso sarà negativo.
' Muovendoci all'interno dello schermo avremo un numero negativo, verso l'esterno dello schermo
' avremo un numero positivo.

' glTranslatef(x, y, z) sposta lungo gli assi X, Y e Z axis, in quest'ordine. La linea di codice qui in
' basso muove a sinistra sull'asse X di 1.5 unità. Non effettua movimenti sull'asse Y (0.0) e muove
' all'interno dello schermo di 6.0 unità. Quando effettuate una traslazione, non la effettuate dal
' centro dello schermo, ma dal punto dove vi trovate in quel momento.

        Gl.Translatef(-1.5, 0.0, -6.0)

' Ora che ci siamo spostati nella metà di sinistra dello schermo e abbiamo settato il punto di vista
' abbastanza in profondità all'interno dello schermo (6.0) da vedere l'intera scena, creeremo un
' triangolo. gl.Begin(GL_TRIANGLES) significa che vogliamo cominciare a tracciare un triangolo
' e gl.End() dice alle OpenGL che abbiamo terminato la creazione del triangolo. Di norma se vi
' servono 3 punti, utilizzate GL_TRIANGLES. Disegnare triangoli è abbastanza veloce in ogni
' scheda video. Se vi servono 4 punti utilizzate GL_QUADS per rendervi semplice la vita. Da quello
' che ho sentito, la maggior parte delle schede video comunque tracciano dei triangoli. Infine, se vi
' servono più di 4 punti, utilizzate GL_POLYGON.

'Nel nostro semplice programma, disegneremo solo un triangolo.Se volessimo disegnare un secondo
' triangolo, dovremmo includere altre 3 linee di codice (3 punti) giusto dopo le prime tre. Tutte e sei
' le linee di codice dovranno trovarsi tra gl.Begin(GL_TRIANGLES) e gl.End(). E' inutile mettere
' un  gl.Begin(GL_TRIANGLES) e un gl.End() per ogni gruppo di 3 punti se stiamo disegnando
' tutti  triangoli. Questo vale anche per i quadrati. Se sapete che dovrete tracciare solo quadrati,
' potete  includere le seconde 4 linee di codice giusto dopo le prime 4 linee.
```

```
' Un poligono d'altro canto  (GL_POLYGON) può essere formato da quanti punti volete, quindi non
' è importante il numero di ' linee avete tra gl.Begin(GL_POLYGON) e gl.End().
'La prima linea dopo gl.Begin, setta il primo punto del nostro poligono .
' Il primo numero dopo  gl.Vertex è per l'asse X, il secondo numero è per l'asse Y e il terzo per l'asse
'Z.
'Quindi, nella prima  linea, non ci muoviamo lungo l'asse X. Ci muoviamo di una unità lungo l'asse
Y, e non ci  muoviamo lungo l'asse Z. Questo ci restituisce il punto in alto del triangolo.
'Il secondo gl.Vertex  muove a sinistra di una unità lungo l'asse X e in basso di una unità lungo
' l'asse Y. Questo ci  restituisce il punto in basso a sinistra del triangolo. Il terzo glVertex muove a '
' destra di una unità, e  in basso di una unità.
' Questo ci dà il punto in basso a destra del triangolo.
' gl.End() dice alle  OpenGL che non ci sono altri punti. Verrà mostrato il triangolo pieno.

        gl.Begin(gl.GL_TRIANGLES)
                gl.Vertex3f(0.0, 1.0, 0.0)
                gl.Vertex3f(-1.0, -1.0, 0.0)
                gl.Vertex3f(1.0, -1.0, 0.0)
        gl.End


 'Ora che abbiamo il triangolo visualizzato nella metà di sinistra dello schermo, abbiamo bisogno di
' muoverci nella metà di destra dello schermo per disegnare il quadrato.
' Per farlo useremo di nuovo gl.Translate.
' Questa volta dobbiamo muoverci verso destra, quindi X deve essere un valore positivo.
' Poiché ci siamo già mossi a sinistra di 1.5 unità, per tornare al centro dobbiamo muoverci a destra
' di 1.5 unità.
' Dopo aver raggiunto il centro dobbiamo muoverci di altre 1.5 unità a destra del centro.
' Quindi in totale dobbiamo muoverci di 3.0 unità verso destra.

          Gl.Translatef(3.0, 0.0, 0.0)

' Ora creiamo il quadrato. Lo faremo utilizzando GL_QUADS.
' Un quadrilatero è fondamentalmente un poligono di 4 lati. Perfeto per fare un quadrato.
'Il codice per creare un quadrato è molto simile al codice usato per creare il triangolo.
' La sola differenza è l'utilizzo di GL_QUADS al posto di GL_TRIANGLES,
' e un ulteriore gl.Vertex3f per il quarto punto del quadrato.
' L'ordine di creazione del quadrato sarà alto-sinistra, alto-destra, basso-destra, basso-sinistra.

        gl.Begin(gl.GL_QUADS)
                gl.Vertex3f(-1.0, 1.0, 0.0)
                gl.Vertex3f(1.0, 1.0, 0.0)
                gl.Vertex3f(1.0, -1.0, 0.0)
                gl.Vertex3f(-1.0, -1.0, 0.0)
        gl.End
```

ed ora chiudiamo la funzione

```
        WAIT 0.01
END
```

lanciamo il programma e comparirà la nostra finestra con il triangolo e il quadrato


tutorial_opengl

Lezione 3

In questa lezione impareremo come aggiungere due diversi tipi di colore al triangolo e al quadrato. Il flat coloring renderà il quadrilatero di un colore pieno. Lo smooth coloring mescolerà assieme i 3 colori specificati per ogni punto (vertice) del triangolo, creando una simpatica sfumatura dei colori.

Utilizzando il codice dell'ultimo tutorial, ne aggiungeremo altro nella procedura Screen_draw() Riscriverò lintera procedura qui in basso, così se avete intenziine di modificare la scorsa lezione, potete sostituire la procedura Screen_draw() con questa, o semplicemente aggiungere il codice alla procedura Screen_draw() il codice che non è presente nell'ultimo tutorial.

```
PUBLIC SUB Screen_draw()

    gl.ClearColor(0, 0, 0, 1)
    Gl.Clear(gl.GL_COLOR_BUFFER_BIT AND gl.GL_DEPTH_BUFFER_BIT)

    Gl.LoadIdentity()
    Gl.Translatef(-1.5, 0.0, -6.0)

    gl.Begin(gl.GL_TRIANGLES)
```

'Se ricordate dall'ultimo tutorial, questa è la sezione di codice per disegnare il triangolo nella metà
'sinistra dello schermo. Nella prossima linea di codice utilizzeremo per la prima volta il comando
'gl.Color3f(r,g,b). I tre parametri nelle parentesi sono i valori dell'intensità del rosso, verde e blu. I
'valori vanno da 0.0f a 1.0f. Funzionano allo stesso modo dei valori del colore per cancellare lo
'sfondo dello schermo.

'Imposteremo il colore in rosso (intensità massima per il rosso, niente verde, niente blu). La linea di
'codice immediatamente successiva a questa è il primo vertice (la parte alta del triangolo), e verrà
'disegnata utilizzando il colore corrente che è il rosso. Qualsiasi cosa disegneremo da adesso in poi
'sarà rossa fino a quando non cambieremo il colore in qualcosa di diverso dal rosso.

```
        gl.Color3f(1.0,0.0,0.0)
        gl.Vertex3f( 0.0, 1.0, 0.0)
```

'Abbiamo posizionato il primo vertice sullo schermo, impostando il suo colore come rosso. Ora,
'prima di posizionare il secondo vertice cambieremo il colore in verde. In questo modo il secondo
'vertice che è l'angolo sinistro del triangolo verrà disegnato verde.

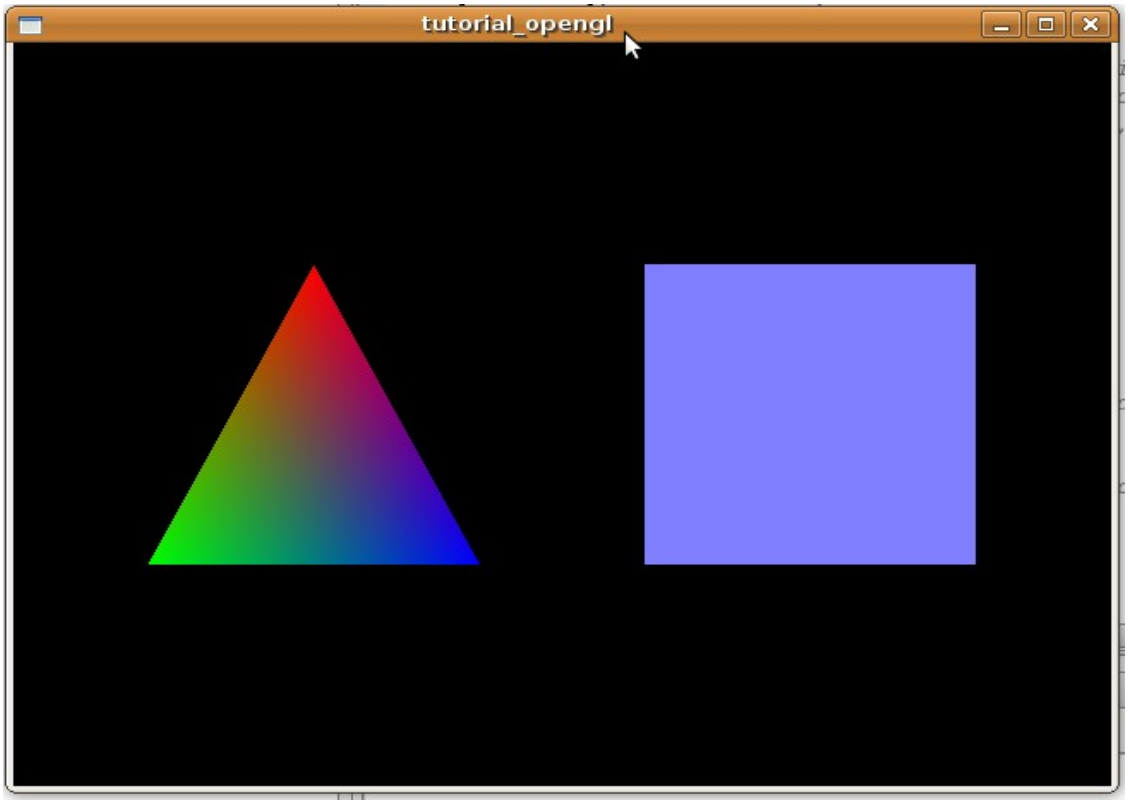
```
        gl.Color3f(0.0, 1.0 , 0.0)
        gl.Vertex3f(-1.0 , -1.0 , 0.0)
```

'Ora siamo sul terzo e ultimo vertice. Appena prima di disegnarlo, setteremo il suo colore a blu.
'Questo sarà l'angolo destro del triangolo.
'Appena il comando gl.End() viene raggiunto, il poligono sarà riempito.
'Ma poiché ci sono colori differenti per ogni vertice, piuttosto che un unico colore solido, il colore si
'espanderà da ogni angolo, incontrandosi alla fine nel centro, dove i colori si mescoleranno assieme.
'questo è lo smooth coloring.

```
        gl.Color3f(0.0 , 0.0 , 1.0)
        gl.Vertex3f( 1.0 , -1.0 ,  0.0)
```

```
        gl.End()
        gl.Translatef(3.0 , 0.0 , 0.0)

'Ora disegneremo un quadrato blu. E' importante ricordare che qualsiasi cosa disegnata dopo che il
'colore è stato impostato verrà disegnata in quel colore. Ogni progetto che create lungo la vostra
'strada utilizzerà o lo smooth o il flat coloring. Perfino in scene dove qualsiasi cosa è ricoperta di
'textures, gl.Color3f può anche essere usato per colorare col colore delle textures, ecc.
' Altro su questo argomento più avanti.

'Quindi per disegnare il nostro quadrato di un unico colore, tutto quello che dobbiamo fare è
'impostare il colore una volta nel colore che vogliamo (blu in questo esempio), quindi disegnare il
'quadrato. Il colore blu verrà usato per ogni vertice perché non c'è niente che dice alle OpenGL di
'cambiare il colore ad ogni vertice. Il risultato finale... un quadrato blu.

        gl.Color3f(0.5 , 0.5 , 1.0)

        gl.Begin(gl.GL_QUADS)
                gl.Vertex3f(-1.0, 1.0, 0.0)
                gl.Vertex3f(1.0, 1.0, 0.0)
                gl.Vertex3f(1.0, -1.0, 0.0)
                gl.Vertex3f(-1.0, -1.0, 0.0)
        gl.End

  WAIT 0.01
END
```

Ora possiamo lanciare il programma e visualizzarne il risultato



Lezione 4

In questa lezione impareremo come ruotare questi oggetti colorati attorno ad un asse.

Utilizzando il codice dell'utlimo tutorial, vi aggiungeremo alcune cose. Riscriverò l'intera sezione di codice qui in basso così vi sarà semplice vedere che cosa andrà aggiunto e cosa andrà sostituito.

Comiceremo con l'aggiungere le due variabili che conterrano la traccia della rotazione per ciascun oggetto.
Lo faremo proprio all'inizio del programma.
Noterete in basso che ho aggiunto due linee dopo PRIVATE Screen .....
Queste linee impostano due variabili floating point che possiamo usare per ruotare gli oggetti con un'accuratezza molto elevata.
I floating point consentono l'utilizzo di numeri decimali.
Ciò significa che non siamo vincolati all'uso di 1, 2, 3 per l'angolo, ma possiamo utilizzare 1.1, 1.7, 2.3, o persino 1.015 per una maggiore precisione.
Scoprirete che i numeri floating point sono essenziali nella programmazione OpenGL.

```
' Gambas module file


PRIVATE CONST ScrWidth AS Integer = 640
PRIVATE CONST ScrHeight AS Integer = 480
PRIVATE Screen AS NEW Window(TRUE) AS "Screen"
PRIVATE rtri AS Float
PRIVATE rquad AS Float
```

Ora abbiamo bisogno di modificare il codice in Screen_draw().
Riscriverò l'intera procedura.
Questo dovrebbe farvi notare con maggior semplicità i cambiamenti che ho apportato al codice originale.
Spiegherò perché le linee sono state modificate e cosa fanno esattamente le nuove linee.
La sezione successiva di codice è esattamente la stessa dell'ultimo tutorial.

```
PUBLIC SUB Screen_draw()


        gl.ClearColor(0, 0, 0, 1)
        Gl.Clear(gl.GL_COLOR_BUFFER_BIT OR  gl.GL_DEPTH_BUFFER_BIT)

        Gl.LoadIdentity()
        Gl.Translatef(-1.5, 0.0, -6.0)
```

'La prossima linea di codice è nuova. gl.Rotatef(Angle,Xvector,Yvector,Zvector) è responsabile
'della rotazione dell'oggetto attorno ad un asse.
'Troverete questo comando estremamente utile.
'Angle è un numero (di norma contenuto in una variabile) che rappresenta quanto volete ruotare
'l'oggetto.
'I parametri Xvector, Yvector e Zvector rappresentano il vettore attorno al quale avrà luogo la
'rotazione.
'Se utilizzate i valori (1,0,0), state descrivendo un vettore che viaggia in una direzione di 1 unità
'lungo l'asse x verso destra.

'I valori (-1,0,0) descrivono un vettore che viaggia nella direzione di 1 unità lungo l'asse x, ma
'questa volta verso sinistra.
'
'D. Michael Traub: ha fornito la seguente spiegazione dei parametri Xvector, Yvector e Zvector.
'
'Per capire meglio le rotazioni di X, Y e Z le spiegherò attraverso esempi...
'
'Asse X - State lavorando ad una sega a banco. La barra che passa per il centro della lama va da
'sinistra a destra (proprio come l'asse x in OpenGL). I denti della lama ruotano attorno all'asse x (la
'barra che passa atraverso il centro della lama), e sembra che tagli verso di voi o nella direzione
'opposta, a seconda della direzione in cui viene spinta la lama. Quando ruotiamo qualcosa sull'asse x
'in OpenGL verrà ruotata allo stesso modo.
'
'Asse Y - Immaginate di stare nel mezzo di un campo. C'è un enorme tornado che viene dritto verso
'di voi. Il centro del tornado va dal cielo verso terra (su e giu, proprio come l'asse y in OpenGL). I
'rottami nel tornado ruotano attorno all'asse y (centro del tornado) da sinistra verso destra o da
'destra verso sinistra. Quando ruotate qualcosa sull'asse y in OpenGL ruota allo stesso modo.

'Asse Z - State guardando la parte anteriore di un ventilatore. Il centro del ventilatore punta verso di
'voi e nella direzione opposta (proprio come l'asse z in OpenGL). Le pale del ventilatore ruotano
'attorno all'asse z (centro del ventilatore) in senso orario o antiorario. Quando ruotate qualcosa
'sull'asse z in OpenGL ruota allo stesso modo.

'Così, nella linea di codice seguente, se rtri è uguale a 7, ruoteremo di 7 sull'asse Z

```
gl.Rotatef(rtri,0.0 , 0.0 , 1.0 )
```

'La sezione di codice successiva non è stata cambiata.
'Disegna un triangolo dai colori sfumati.
'Il triangolo verrà disegnato nella metà di sinistra dello schermo e verrà ruotato sul suo asse Z,

```
gl.Begin(gl.GL_TRIANGLES)

        gl.Color3f(1.0, 0.0, 0.0)
        gl.Vertex3f(0.0, 1.0, 0.0)
        gl.Color3f(0.0, 1.0, 0.0)
        gl.Vertex3f(-1.0, -1.0, 0.0)
        gl.Color3f(0.0, 0.0, 1.0)
        gl.Vertex3f(1.0, -1.0, 0.0)

gl.End()
```

'Noterete nel codice seguente che abbiamo aggiunto un altro gl.LoadIdentity().
' Lo abbiamo messo per azzerare la visuale.
'Se non si effettua il reset della visuale, se si trasla un oggetto dopo che è stato ruotato,
'si potrebbero avere risultati veramente inaspettati.
'Poiché l'asse è stato ruotato, potrebbe non puntare nella direzione che pensate.
'Quindi se trasliamo verso sinistra sull'asse X, potremmo finire per effettuare un movimento verso
'l'alto o verso il basso, a seconda di quanto abbiamo ruotato su ogni asse.

'Provate a togliere gl.LoadIdentity() per vedere cosa succede.

'Una volta che la scena è stata azzerata in modo che X va da sinistra a destra, Y verso l'alto e verso 'il basso e Z verso dentro e verso fuori, possiamo translare.
'Noterete che ci stiamo muovendo solo di1.5 verso destra invece di 3.0 come abbiamo fatto nell'ultima lezione.
'Quando azzeriamo lo schermo, il nostro fuoco si muove verso il centro dello schermo.
'Ciò significa che non siamo più 1.5 unità verso destra, ma siamo tornati a 0.0.
'Quindi per muoverci di 1.5 a destra di zero non dobbiamo muoverci di 1.5 da sinistra verso il 'centro, quindi di 1.5 verso destra (totale 3.0); dobbiamo semplicemente muoverci dal centro verso 'destra di sole 1.5 unità.

'Dopo che ci siamo spostati nel nuovo punto nella parte destra dello schermo, ruotiamo il quadrato 'sull'asse Z.

```
gl.LoadIdentity()
gl.Translatef(1.5 , 0.0 , -6.0)
gl.Rotatef(rquad,1.0 , 0.0 , 0.0 )
```

'Questa sezione di codice rimane la stessa.
'Disegna un quadrato blu a partire da un quadrilatero.
'Disegnerà il quadrato nella parte destra dello schermo.

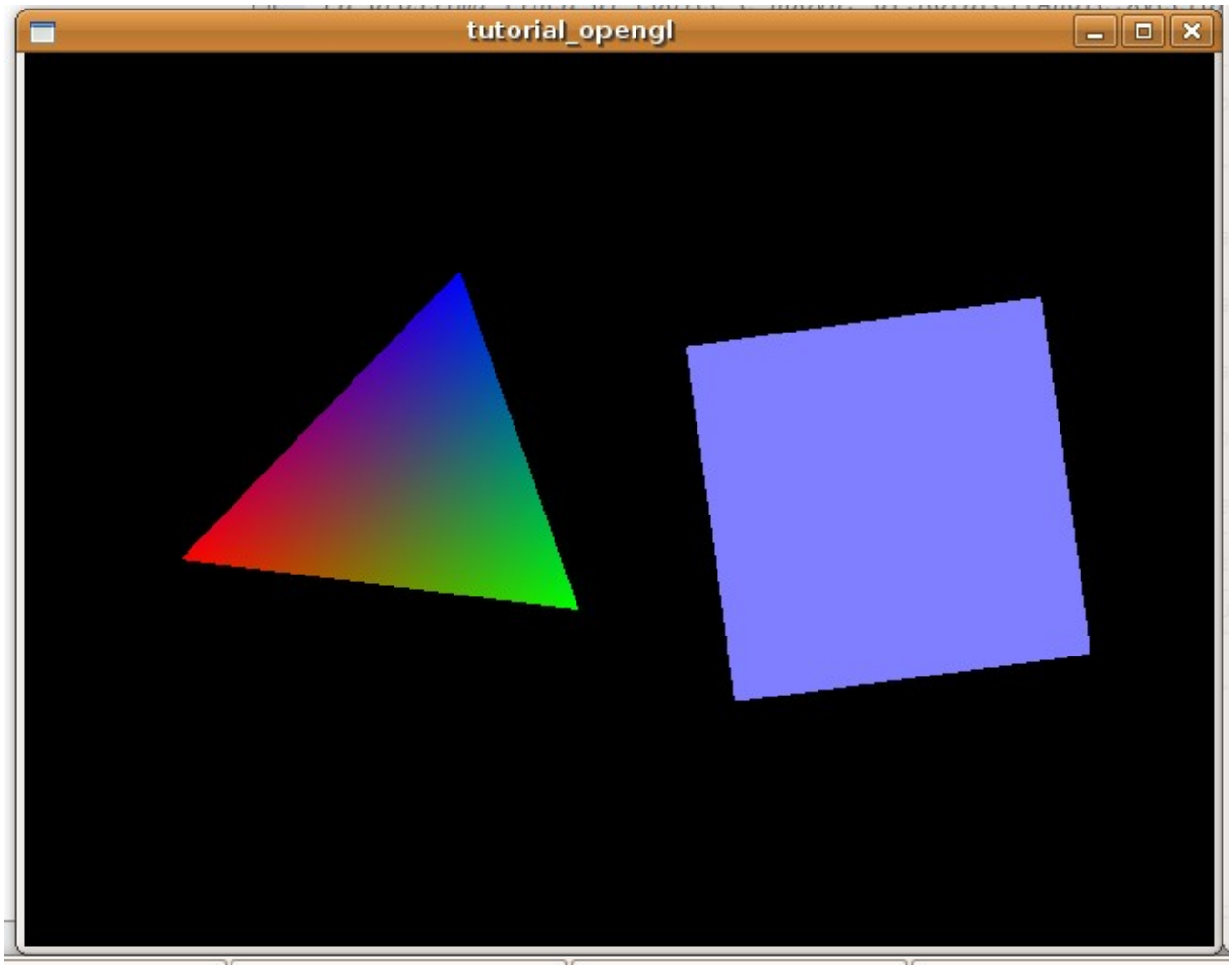
```
gl.Color3f(0.5, 0.5, 1.0)

gl.Begin(gl.GL_QUADS)
        gl.Vertex3f(-1.0, 1.0, 0.0)
        gl.Vertex3f(1.0, 1.0, 0.0)
        gl.Vertex3f(1.0, -1.0, 0.0)
        gl.Vertex3f(-1.0, -1.0, 0.0)
 gl.End
```

'Le prossime due linee sono nuove.
'Pensate a rtri e rquad come contenitori. All'inizio del nostro programma li abbiamo dichiarati .
'Quando costruiamo questi contenitori, essi non contengono niente.
'La prima linea in basso AGGIUNGE 0.2 al contenitore.
'Quindi ogni volta che controlliamo il valore del contenitore rtri dopo questa sezione di codice, esso 'risulterà incrementato di 0.2.
'il contenitore rquad viene decrementato di 0.15.
'Quindi ogni volta che controlliamo il valore del contenitore rquad, esso risulterà decrementato di '0.15. Il decrescere causerà la rotazione dell'oggetto nella direzione opposta che avrebbe se il valore 'crescesse.

'Provate a cambiare + in - nella linea successiva, per vedere come gli oggetti ruotano nelle direzioni 'inverse.
'Provate a cambiare i valori da 0.2 a 1.0. Più alto è il numero, più veloce sarà la rotazione.
'Più è basso, più sarà lenta.

```
        rtri += 0.2
        rquad -= 0.15
WAIT 0.01
END
```

ora possiamo lanciare l' applicazione e visualizzare il risultato



Lezione 5

Espandendo l'ultimo tutorial, costruiremo ora degli oggetti in VERO 3D, piuttosto che oggetti 2D in un mondo 3D.
Faremo ciò aggiungendo al triangolo un lato sinistro, un lato destro e un lato posteriore ed un lato sinistro, uno destro, uno posteriore, uno superiore e uno inferiore al quadrato.
Facendo questo, trasformeremo il triangolo in una piramide e il quadrato in un cubo.

Sfumeremo i colori della piramide, creando un oggetto in smooth color, e per il cubo coloreremo ogni faccia con un colore diverso.

```
PUBLIC SUB Screen_draw()

        gl.ClearColor(0, 0, 0, 0)
        Gl.Clear(gl.GL_COLOR_BUFFER_BIT OR gl.GL_DEPTH_BUFFER_BIT)

        Gl.LoadIdentity()
        Gl.Translatef(-1.5, 0.0, -6.0)
        Gl.Rotatef(0, 1.0, 0.0, 0.0)
        Gl.Rotatef(rtri, 0.0, 1.0, 0.0)
        Gl.Rotatef(0, 0.0, 0.0, 1.0)

   gl.Begin(gl.GL_TRIANGLES)

'Alcuni di voi hanno preso il codice dall'ultimo tutorial ed hanno costruito degli oggetti 3D per
'conto loro.
' Una cosa che succede spesso è che gli oggetti non ruotano sul loro asse, ma sembra come che
'ruotino su tutta la superficie dello schermo.
'Per far ruotare un oggetto su un asse, occorre che venga disegnato ATTORNO a questo asse.
'Dovete ricordare che il centro di un oggetto qualsiasi dovrebbe essere 0 sulla X, 0 sulla Y e 0 sulla
'Z.

'Il codice seguente creerà una piramide attorno all'asse centrale.
'La parte superiore della piramide si trova una unità sopra il centro, la parte inferiore della piramide
'si trova una unità sotto il centro.
' Il punto in alto si trova proprio al centro (zero), e i punti in basso si trovano uno alla sinistra del
'centro e uno alla destra.

'Notare che tutti i triangoli vengono tracciati in senso orario.
'Questa cosa è importante e verrà spiegata in un tutorial futuro.
'Per adesso, sappiate solo che è buona norma costruire oggetti o in senso orario o in senso
'antiorario, ma non si dovrebbero mischiare i due modi senza una valida ragione.

'Cominceremo col tracciare la faccia anteriore.
'Poiché tutte le facce condividono il punto in alto, lo faremo rosso per tutti i triangoli. Il colore dei
'due punti in basso dei triangoli sarà alternato.
'La faccia anteriore avrà il punto di sinistra verde e quello di destra blu.
'Quindi il triangolo sul lato destro avrà il punto di sinistra blu e quello di destra verde.
'Alternando i due colori in basso di ogni faccia, avremo un punto comune dello stesso colore in
'basso per ogni faccia.
                gl.Color3f(1.0, 0.0, 0.0)
```

```
            gl.Vertex3f(0.0, 1.0, 0.0)
            gl.Color3f(0.0, 1.0, 0.0)
            gl.Vertex3f(-1.0, -1.0, 1.0)
            gl.Color3f(0.0, 0.0, 1.0)
            gl.Vertex3f(1.0, -1.0, 1.0)

'Ora disegnamo la faccia di destra.
'Notare che i due punti in basso vengono disegnati una unità a destra del centro, e il punto in alto
'viene disegnato una unità sull'asse y e proprio al centro dell'asse x, causando la pendenza della
f'accia dal punto centrale in alto verso la parte destra dello schermo in basso.

'Notare che il punto di sinistra questa volta verrà disegnato blu.
'Disegnandolo blu, avrà lo stesso colore dell'angolo in basso a destra della faccia anteriore,
'mescolando il blu da quest'angolo verso la faccia anteriore e di destra della piramide.

'Da notare anche come le rimanenti tre facce siano incluse nello stesso gl.Begin(GL_TRIANGLES)
'e gl.End() come la prima faccia.
'Poiché stiamo facendo quest'oggetto interamente con triangoli, OpenGL saprà che ogni serie di tre
'punti disegnamo sono i tre punti di un triangolo.
'Una volta disegnati tre punti, se ce ne sono altri tre, riterrà che dovrà essere tracciato un altro
'triangolo.
'Se mettessimo quattro punti invece di tre, OpenGL traccerebbe i primi tre e presumerebbe che il
'quarto fosse il punto di partenza di un nuovo triangolo.
'Non verrebbe disegnato un quadrato.
'Quindi assicuratevi di non aggiungere per sbaglio nessun punto ulteriore.

             gl.Color3f(1.0, 0.0, 0.0)
            gl.Vertex3f(0.0, 1.0, 0.0)
            gl.Color3f(0.0, 0.0, 1.0)
            gl.Vertex3f(1.0, -1.0, 1.0)
            gl.Color3f(0.0, 1.0, 0.0)
            gl.Vertex3f(1.0, -1.0, -1.0)

'Adesso la faccia posteriore.
'Ancora una volta cambia il colore.
'Il punto di sinistra è adesso ancora verde, poiché l'angolo che condivide con la faccia destra è
'verde.

             gl.Color3f(1.0, 0.0, 0.0)
             gl.Vertex3f(0.0, 1.0, 0.0)
             gl.Color3f(0.0, 1.0, 0.0)
             gl.Vertex3f(1.0, -1.0, -1.0)
             gl.Color3f(0.0, 0.0, 1.0)
             gl.Vertex3f(-1.0, -1.0, -1.0)


'Infine disegnamo la faccia di sinistra. I colori cambieranno un'ultima volta.
'Il punto a sinistra è blu e si mescola con il punto a destra della faccia posteriore.
'Il punto di sinistra è verde e si mescola con il punto di sinistra della faccia anteriore.

'Abbiamo finito il disegno della piramide. poiché la piramide ruota solo sull'asse Y, non vedremo
' mai il basso, quindi non c'è bisogno di mettere un fondo alla piramide. Se volete sperimentare,
```

' provate ad aggiungere un fondo utilizzando un quadrato, quindi rotate il tutto lungo l'asse X per
' vedere se lo avete fatto correttamente. Assicuratevi che il colore utilizzato in ogni angolo del
' quadrato corrisponda con i colori usati per i quattro lati della piramide.

```
                gl.Color3f(1.0, 0.0, 0.0) '; / / Red
                gl.Vertex3f(0.0, 1.0, 0.0) '; / / Top OF Triangle(Left)
                gl.Color3f(0.0, 0.0, 1.0) '; / / Blue
                gl.Vertex3f(-1.0, -1.0, -1.0) '; / / Left OF Triangle(Left)
                gl.Color3f(0.0, 1.0, 0.0) '; / / Green
                gl.Vertex3f(-1.0, -1.0, 1.0) ';

        gl.End
```

'Ora disegneremo il cubo.
'Esso è formato da sei quadrati.
' Tutti i quadrati sono disegnati in senso antiorario. Ciò significa che il primo punto è quello in alto
'a destra, il secondo punto è in alto a sinistra, il terzo punto è in basso a sinistra e il quarto punto è in
'basso a destra.
'Quando disegnamo la faccia posteriore, potrebbe sembrare che la stiamo disegnando in senso
'orario, ma dovete tenere presente che se ci troviamo dietro al cubo, guardando verso di esso, la
'parte sinistra dello schermo sarà in realtà la parte destra del quadrato e la parte destra dello schermo
'sarà la in realtà la parte sinistra del quadrato.

'Notare che in questa lezione muoveremo il cubo un pochino verso l'interno dello schermo.
'Facendo questo, la dimensione del cubo apparirà più vicina a quella della piramide.
'Se lo muovete solo di 6 unità nello schermo, il cubo apparirà molto più grande della piramide, e'
'parte di esso potrebbe fuoriuscire dai limiti dello schermo.
'Potete giocherellare con questi settaggi per vedere come muovere il cubo ulteriormente all'interno
'dello schermo lo renda sempre più piccolo, e muoverlo più vicino a noi lo renda più grande.
'Questo accade a causa della prospettiva. Oggetti più lontani appaiono più piccoli :)

```
         gl.LoadIdentity()

        gl.Translatef(1.5, 0.0, -7.0)
        gl.Rotatef(rquad, 1.0, 1.0, 1.0)

        gl.Begin(gl.GL_QUADS)
```

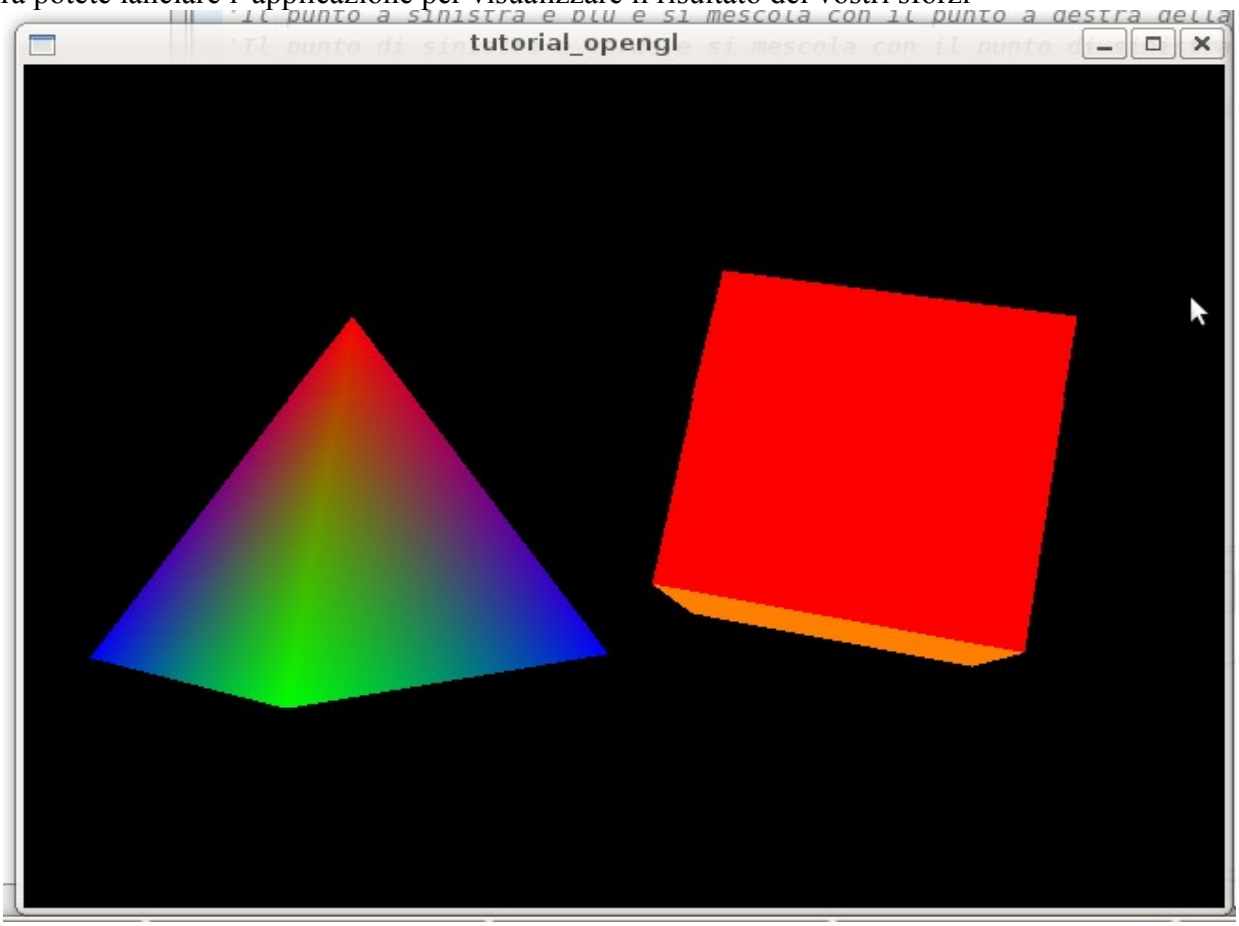
'Cominceremo col disegnare la parte superiore del cubo. Ci muoveremo una unità verso l'alto dal
centro del cubo. Notare che l'asse Y è sempre uno. Disegneremo quindi un quadrato sul piano ,
'vale a dire all'interno dello schermo.
'Cominceremo col disegnare il punto in alto a destra della parte alta del cubo. il punto in alto a
destra dovrebbe essere una unità a destra e una all'interno dello 'schermo.
'Il secondo punto dovrebbe trovarsi una unità a sinistra e una all'interno dello schermo.
'Ora dobbiamo tracciare la parte bassa del quadrato in direzione dell'osservatore.
'Quindi, per fare ciò, al posto di andare all'interno dello schermo, ci muoveremo di una unità verso
lo schermo. ha significato?

```
                gl.Color3f(0.0, 1.0, 0.0)
                gl.Vertex3f(1.0, 1.0, -1.0)
                gl.Vertex3f(-1.0, 1.0, -1.0)
                gl.Vertex3f(-1.0, 1.0, 1.0)
```

```
                gl.Vertex3f(1.0, 1.0, 1.0)

'La parte in basso è disegnata alla stessa maniera di quella in alto, ma poiché è la parte inferiore,
' viene disegnata a partire da una unità in basso dal centro del cubo.
'Notare che l'asse Y è sempre meno uno.
'Se ci troviamo sotto il cubo, guardando verso il quadrato che forma il fondo, noteremo che l'angolo
' in alto a destra è l'angolo più vicino all'osservatore, quindi al posto di disegnare prima verso la
' distanza, disegneremo per prima la parte più vicina all'osservatore, quindi il lato sinistro più vicino
' all'osservatore, quindi andremo verso l'interno dello schermo per disegnare i due punti inferiori.

'Se non volete davvero preoccuparvi dell'ordine in cui vengono disegnati i poligoni (orario o meno),
' potete semplicemente copiare lo stesso codice del quadrato superiore, muoverlo in basso sull'asse
' Y a -1 e funzionerà, ma ignorare l'ordine in cui viene disegnato il quadrato potrebbe causare
' risultati inattesi una volta che andrete ad utilizzare cose simpatiche, tipo texture mapping.

                 gl.Color3f(1.0, 0.5, 0.0)
                gl.Vertex3f(1.0, -1.0, 1.0)
                gl.Vertex3f(-1.0, -1.0, 1.0)
                gl.Vertex3f(-1.0, -1.0, -1.0)
                gl.Vertex3f(1.0, -1.0, -1.0)

'Ora disegneremo il frontale del quadrato.
'Ci muoviamo una unità in direzione dello schermo e lontano dal centro per disegnare la faccia
' frontale.
'Notare che l'asse Z è sempre uno. Nella piramide l'asse Z non era sempre uno. In alto, l'asse Z era
'zero.
'Se provate a cambiare l'asse Z in zero nel codice seguente, potrete notare che l'angolo che avete
' cambiato si inclina verso l'interno dello schermo.
' Questa è una cosa che per il momento non ci interessa fare :)

                 gl.Color3f(1.0, 0.0, 0.0)
                gl.Vertex3f(1.0, 1.0, 1.0)
                gl.Vertex3f(-1.0, 1.0, 1.0)
                gl.Vertex3f(-1.0, -1.0, 1.0)
                gl.Vertex3f(1.0, -1.0, 1.0)

'La faccia posteriore è un quadrato proprio come la faccia anteriore, ma è spostata verso l'interno
' dello schermo.
'Notare che l'asse Z è ora meno uno per tutti i punti.

                gl.Color3f(1.0, 1.0, 0.0)
                gl.Vertex3f(1.0, -1.0, -1.0)
                gl.Vertex3f(-1.0, -1.0, -1.0)
                gl.Vertex3f(-1.0, 1.0, -1.0)
                gl.Vertex3f(1.0, 1.0, -1.0)

'Ora dobbiamo solo disegnare altri due quadrati e avremo finito.
'Come al solito, potete notare che un asse è sempre lo stesso per tutti i punti. In questo caso l'asse X
'è sempre meno uno.
'Questo perché stiamo disegnando sempre alla sinistra del centro poiché è la faccia sinistra.

                gl.Color3f(0.0, 0.0, 1.0)
```

```
                gl.Vertex3f(-1.0, 1.0, 1.0)
                gl.Vertex3f(-1.0, 1.0, -1.0)
                gl.Vertex3f(-1.0, -1.0, -1.0)
                gl.Vertex3f(-1.0, -1.0, 1.0)

'Questa è l'ultima faccia per completare il cubo.
' L'asse X è sempre uno. Disegneremo in senso antiorario.
'Se volete, potete ignorare quasta faccia e fare una scatola :)

'Oppure, se volete sperimentare, potreste provare a cambiare il colore di ogni punto per sfumarlo
'allo stesso modo della piramide.

                gl.Color3f(1.0, 0.0, 1.0) '; / / Set The Color TO Violet
                gl.Vertex3f(1.0, 1.0, -1.0) '; / / Top Right OF The Quad(Right)
                gl.Vertex3f(1.0, 1.0, 1.0) '; / / Top Left OF The Quad(Right)
                gl.Vertex3f(1.0, -1.0, 1.0) '; / / Bottom Left OF The Quad(Right)
               gl.Vertex3f(1.0, -1.0, -1.0) ';

          gl.End
           rtri -= 02
           rquad += 0.8

         WAIT 0.01
END
```

Ora potete lanciare l' applicazione per visualizzare il risultato dei vostri sforzi



Lezione 6

Imparare come mappare le textures porta molti vantaggi.
Immaginiamo di volere un missile che vola attraverso lo schermo. Fino a questo tutorial avremmo probabilmente costruito l'intero missile di poligoni e colori fantasiosi. Con il texture mapping, potete prendere una fotografia vera di un missile e farla volare per lo schermo.
Quale pensate che sarà migliore? Una fotografia o un oggetto formato da triangoli e quadrati?
Utilizzando il texture mapping, non solo apparirà migliore, ma il vostro programma girerà anche più velocemente. Il missile texture mapped sarà solo un quadrato che si muove sullo schermo. Un missile fatto di poligoni potrebbe essere formato da centinaia o migliaia di poligoni.
Un singolo quadrato texture mapped utilizzerà molta meno potenza elaborativa.

Cominciamo aggiungendo cinqe nuove linee di codice alle dichiarazioni di variabili globali
La prima nuova linea è PRIVATE texture AS NEW Integer[] in cui dichiariamo una variabile array integer che conterra l' indice delle texture. Le altre tre righe sono le variabili che useremo per settare la velocità di rotazione del cubo sui vari assi.

```
' Gambas module file

PRIVATE CONST ScrWidth AS Integer = 640
PRIVATE CONST ScrHeight AS Integer = 480
PRIVATE Screen AS NEW Window(TRUE) AS "Screen"
PRIVATE texture AS NEW Integer[]
PRIVATE xrot AS Float
PRIVATE yrot AS Float
PRIVATE zrot AS Float
PRIVATE logo AS image
```

Ora , dobbiamo aggiungere un codice nella funzione di inizializzazione delle openg l che ci consenta di gestire le texture quindi la funzione InitGL() sarà così modificata

```
PUBLIC SUB InitGL()

        Gl.ShadeModel(gl.GL_SMOOTH)
        gl.ClearColor(0.0, 0.0, 0.5, 0.5)
        Gl.Clear(gl.GL_COLOR_BUFFER_BIT AND gl.GL_DEPTH_BUFFER_BIT)
        Gl.ClearDepth(1.0)
        Gl.Enable(gl.GL_DEPTH_TEST)
          ' Enable texturing
        Gl.Enable(Gl.GL_TEXTURE_2D)
        Gl.DepthFunc(gl.GL_LEQUAL)
        Gl.Hint(gl.GL_PERSPECTIVE_CORRECTION_HINT, gl.GL_NICEST)
      LoadTextures()
END
```

Si noti che abbiamo aggiunto anche la chiamata ad una funzione LoadTextures() in cui ora vi scriveremo il codice per caricare la texture che ci servirà:

```
PUBLIC SUB LoadTextures()

        logo = image.Load("pippo.png")
        texture = Gl.GenTextures(1)
        Gl.BindTexture(gl.GL_TEXTURE_2D, texture[0])
       Gl.TexImage2D(logo)


END

'la variabile logo di tipo image la useremo per caricare un immagine (in questo caso pippo.png) che
'trasformeremo in texture  con gl.TxImage2D(logo)

'ci sono alcune cose MOLTO importanti che dovete sapere sulle immagini che avete intenzione di
'usare come textures. L'altezza e la larghezza dell'immagine DEVE essere un multiplo di 2. Altezza
'e larghezza devono essere di almeno 64 pixels e, per motivi di compatibilità, non devono essere
'maggiori di 256 pixels. Se l'immagine che avete intenzione di utilizzare non è di 64, 128 o 256
'pixels in larghezza o in altezza, ridimensionatela con un programma di disegno. Esistono dei
'metodi per aggirare questa limitazione, ma per ora ci occuperemo solamente delle textures di
'dimensioni standard.

' Le prossime due linee di codice dicono alle OpenGL quale tipo di filtering utilizzare quando
'l'immagine è più larga (GL_TEXTURE_MAG_FILTER) o allungata sullo schermo rispetto alla
'texture originale, o quando è più piccola (GL_TEXTURE_MIN_FILTER) della texture che ci
' serve. Di norma utilizzo GL_LINEAR per tutti e due i casi. Questo rende la texture sfumata alla
'distanza e quando è vicina allo schermo. L'utilizzo di GL_LINEAR richiede un gran lavoro da
'parte del processore e della scheda video, quindi se il vostro sistema è lento, dovreste utilizzare
'GL_NEAREST. Una texture filtrata tramite GL_NEAREST apparirà pixellosa se stretchata. Potete
'anche provare una combinazione di entrambi i metodi. Provate a filtrare le textures da vicino e a
'non filtrarle da lontano.

 Gl.TexParameteri(gl.GL_TEXTURE_2D, Gl.GL_TEXTURE_MIN_FILTER, gl.GL_LINEAR)
  Gl.TexParameteri(gl.GL_TEXTURE_2D, gl.GL_TEXTURE_MAG_FILTER, gl.GL_LINEAR)
END

'Ora disegnamo il cubo texturizzato.
'Potete sostituire il codice di Screen_draw() con quello qui sotto, oppure potete aggiungere il nuovo
'codice al codice della lezione uno. Questa sezione sarà ultra commentata per farvi comprendere
'meglio.
' Le prime due linee di codice gl.Clear() e gl.LoadIdentity() sono nel codice della lezione uno.
'gl.Clear(GL_COLOR_BUFFER_BIT  OR GL_DEPTH_BUFFER_BIT) cancellerà lo schermo col
'colore selezionato in InitGL().
' In questo caso, lo schermo verrà cancellato col blu. Verrà cancellato anche il depth buffer. Verrà
'quindi azzerata la visuale tramite gl.LoadIdentity().

PUBLIC SUB Screen_draw()

        Gl.Clear(gl.GL_COLOR_BUFFER_BIT OR gl.GL_DEPTH_BUFFER_BIT)
        Gl.LoadIdentity()
       Gl.Translatef(0.0, 0.0, -5.0)

'Le seguenti tre linee di codice ruoteranno il cubo sull'asse x, poi l'asse y ed infine l'asse z. Quanto
```

'ruota su ogni asse dipenderà dai valori memorizzati in xrot, in yrot e in zrot.

```
g l.Rotatef(xrot, 1.0, 0.0, 0.0)
gl.Rotatef(yrot, 0.0, 1.0, 0.0)
gl.Rotatef(zrot, 0.0, 0.0, 1.0)
```

'La linea seguente di codice seleziona quale texture vogliamo utilizzare.
' Se volete utilizzare più di una texture nella vostra scena, dovete selezionare la texture usando
'glBindTexture(GL_TEXTURE_2D, texture[numero della texture da utilizzare]). Se desiderate
' cambiare le textures, potete selezionarne una nuova. Una cosa da notare è che non potete cambiare
'texture all'interno di glBegin() e glEnd(); bisogna farlo prima o dopo glBegin().
' Notare come utilizziamo glBindTextures per specificare quale texture generare e per selezionare
'una texture specifica.

```
glBindTexture(GL_TEXTURE_2D, texture[0])
```

'Per mappare correttamente una texture su un quadrato, dovete assicurarvi che la parte superiore
'destra della texture venga mappata nella parte superiore destra del quadrato.
'La parte superiore sinistra della texture è mappata nella parte superiore sinistra del quadrato, la
'parte inferiore destra della texture è mappata nella parte inferiore destra del quadrato e, per
'concludere, la parte inferiore sinistra della texture è mappata nella parte inferiore sinistra del
'quadrato.
' Se glii angoli della texture non corrispondono agli stessi angoli del quadrato, l' immagine potrebbe
'sembrare capovolta, obliqua, o addirittura potrebbe non essere visualizzata.

'Il primo valore di gl.TexCoord2f è la coordinata X. 0.0 è la parte sinistra della texture. 0.5 è il
'centro della texture e 1.0 è la parte destra della texture. Il secondo valore di gl.TexCoord2f è la
' coordinata Y. 0.0 è la parte inferiore della texture. 0.5f è il centro della texture e 1.0f è la parte
'superiore della texture.

'Così ora sappiamo che la coordinata superiore di sinistra di una texture è 0.0f sulla X e 1.0 su Y, ed
'il vertice superiore di sinistra di un quadrato è -1.0 sulla X e 1.0 sulla Y.
' Ora tutto quello che dobbiamo fare è abbinare le coordinate delle altre tre texture ai tre angoli
'restanti del quadrato.

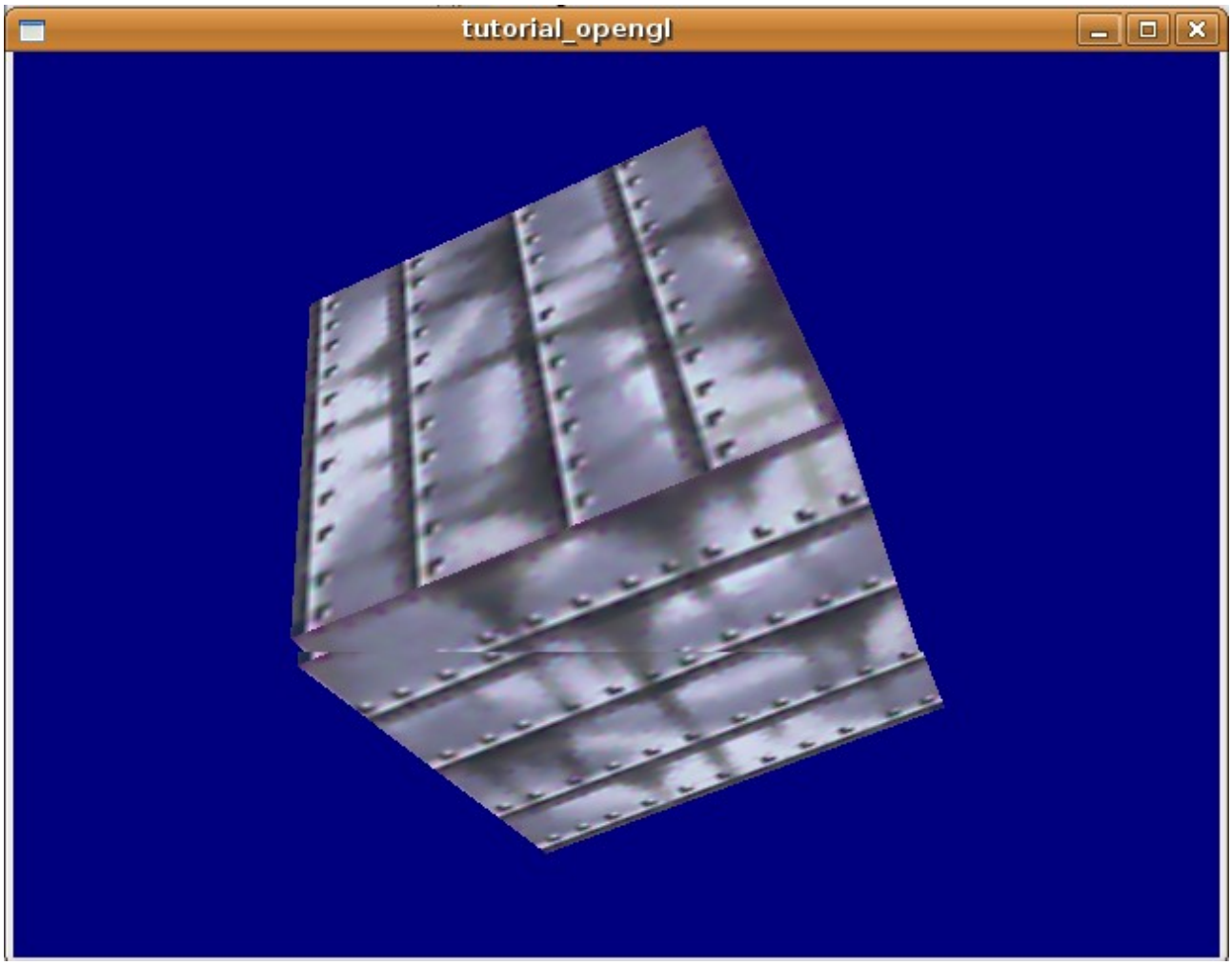
'Provare cambiare i valori y e x di gl.TexCoord2f. Cambiare 1.0 in 0.5fdisegnerà soltanto la metà di
'sinistra della texture da 0.0(sinistra) a 0.5 (centro della texture). Cambiare 0.0 in 0.5 disegnerà
'soltanto la metà di destra di una texture da 0.5f (centro) a 1.0f (destra).

```
gl.Begin(gl.GL_QUADS)

'    / / Front Face
    gl.TexCoord2f(0.0, 0.0)
    gl.Vertex3f(-1.0, -1.0, 1.0) '; / / Bottom Left OF The Texture AND Quad
    gl.TexCoord2f(1.0, 0.0)
    gl.Vertex3f(1.0, -1.0, 1.0) '; / / Bottom Right OF The Texture AND Quad
    gl.TexCoord2f(1.0, 1.0)
    gl.Vertex3f(1.0, 1.0, 1.0) '; / / Top Right OF The Texture AND Quad
    gl.TexCoord2f(0.0, 1.0)
    gl.Vertex3f(-1.0, 1.0, 1.0) '; / / Top Left OF The Texture AND Quad
```

```
 '  / / Back Face
   gl.TexCoord2f(1.0, 0.0) '
   gl.Vertex3f(-1.0, -1.0, -1.0) '; / / Bottom Right OF The Texture AND Quad
   gl.TexCoord2f(1.0, 1.0) '
   gl.Vertex3f(-1.0, 1.0, -1.0) '; / / Top Right OF The Texture AND Quad
   gl.TexCoord2f(0.0, 1.0)
   gl.Vertex3f(1.0, 1.0, -1.0)
   gl.TexCoord2f(0.0, 0.0)
   gl.Vertex3f(1.0, -1.0, -1.0) '; / / Bottom Left OF The Texture AND Quad
 "  / / Top Face
   gl.TexCoord2f(0.0, 1.0)
   gl.Vertex3f(-1.0, 1.0, -1.0) '; / / Top Left OF The Texture AND Quad
   gl.TexCoord2f(0.0, 0.0)
   gl.Vertex3f(-1.0, 1.0, 1.0) '; / / Bottom Left OF The Texture AND Quad
   gl.TexCoord2f(1.0, 0.0)
   gl.Vertex3f(1.0, 1.0, 1.0) '; / / Bottom Right OF The Texture AND Quad
   gl.TexCoord2f(1.0, 1.0)
   gl.Vertex3f(1.0, 1.0, -1.0) '; / / Top Right OF The Texture AND Quad
'   / / Bottom Face
   gl.TexCoord2f(1.0, 1.0)
   gl.Vertex3f(-1.0, -1.0, -1.0) '; / / Top Right OF The Texture AND Quad
   gl.TexCoord2f(0.0, 1.0)
   gl.Vertex3f(1.0, -1.0, -1.0) '; / / Top Left OF The Texture AND Quad
   gl.TexCoord2f(0.0, 0.0)
   gl.Vertex3f(1.0, -1.0, 1.0) '; / / Bottom Left OF The Texture AND Quad
   gl.TexCoord2f(1.0, 0.0)
   gl.Vertex3f(-1.0, -1.0, 1.0) '; / / Bottom Right OF The Texture AND Quad
'   / / Right face
   gl.TexCoord2f(1.0, 0.0)
   gl.Vertex3f(1.0, -1.0, -1.0) '; / / Bottom Right OF The Texture AND Quad
   gl.TexCoord2f(1.0, 1.0)
   gl.Vertex3f(1.0, 1.0, -1.0) '; / / Top Right OF The Texture AND Quad
   gl.TexCoord2f(0.0, 1.0)
   gl.Vertex3f(1.0, 1.0, 1.0) '; / / Top Left OF The Texture AND Quad
   gl.TexCoord2f(0.0, 0.0)
   gl.Vertex3f(1.0, -1.0, 1.0) '; / / Bottom Left OF The Texture AND Quad
'   / / Left Face
   gl.TexCoord2f(0.0, 0.0)
   gl.Vertex3f(-1.0, -1.0, -1.0) '; / / Bottom Left OF The Texture AND Quad
   gl.TexCoord2f(1.0, 0.0)
   gl.Vertex3f(-1.0, -1.0, 1.0) '; / / Bottom Right OF The Texture AND Quad
   gl.TexCoord2f(1.0, 1.0)
   gl.Vertex3f(-1.0, 1.0, 1.0) '; / / Top Right OF The Texture AND Quad
   gl.TexCoord2f(0.0, 1.0)
   gl.Vertex3f(-1.0, 1.0, -1.0)

  gl.End

'Ora incrementiamo il valore di xrot, di yrot e di zrot.
' Provate a cambiare il numero di incremento di ogni variabile per rendere la rotazione del cubo più
'veloce o più lenta, o provate a cambiare + in - per far ruotare il cubo nell'altro verso.
```

```
 xrot += 0.8
yrot += 0.8
zrot += 0.8


  WAIT 0.01
END

'Dovreste ora avere una comprensione migliore del texture mapping.
'Dovreste essere in grado di texturare la superficie di ogni quadrato con un' immagine di vostra
'scelta.
'Una volta presa confidenza con il texture mapping 2d, provate ad aggiungere sei textures differenti
'al cubo.

'Il texture mapping non è difficile da capire una volta comprese le coordinate delle textures.
```

Ora lanciate la vostra applicazione per visualizzare il vostro cubo texturizzato



Lezione 7

In questo lezione impareremo come utilizzare tre differenti filtri per le textures.
Impareremo come spostare un oggetto con i tasti della tastiera e come applicare semplici illuminazioni alla vostra scena OpenGL.
Molte cose verranno trattate in questo tutorial, quindi se i tutorials precedenti vi stanno creando dei problemi, tornate indietro e riguardateveli.
È importante avere una buona comprensione dei principi fondamentali prima di passare al codice seguente.

Stiamo andando modificare ancora il codice dalla lezione uno. Come di consueto, se ci sono dei cambiamenti importanti, scriverò l' intera sezione del codice che è stato modificato. Cominceremo aggiungendo alcune nuove variabili al programma.

```
' Gambas module file

PRIVATE CONST ScrWidth AS Integer = 640
PRIVATE CONST ScrHeight AS Integer = 480
PRIVATE Screen AS NEW Window(TRUE) AS "Screen"
PRIVATE texture AS NEW Integer[]
PRIVATE logo AS image

'Le righe qui sotto sono nuove. Stiamo aggiungendo tre variabili boolean.
'Significa che la variabile può soltanto essere TRUE o FALSE. Creiamo una variabile chiamata
'light per tenere conto se l'illuminazione è accesa o spenta. Le variabili lp e fp sono usate per
'memorizzare se i tasti di 'F3' o 'F2' sono stati premuti. Spiegherò più avanti perchè abbiamo
bisogno 'di queste variabili. Per ora, sappiate solo che sono importanti


PRIVATE light AS Boolean
PRIVATE lp AS Boolean
PRIVATE fp AS Boolean

'Ora imposteremo cinque variabili che gestiranno l'angolo sull'asse x (xrot), l'angolo sull'asse y
'(yrot), la velocità con cui la cassa sta ruotando sull'asse x (xspeed) e la velocità con cui la cassa sta
'ruotando sull'asse y (yspeed). Creeremo anche una variabile chiamata z che gestirà quanto in'
'profondità nello schermo (sull' asse z) la cassa si trova.

PRIVATE xrot AS Float = 0.1
PRIVATE yrot AS Float =0.1
PRIVATE z AS Float = 5.0
PRIVATE xspeed AS Float
PRIVATE yspeed AS Float

'Ora creiamo gli arrays che saranno usati per creare l'illuminazione. Useremo due tipi differenti di
'luci. Il primo tipo di luce è chiamato luce ambientale.
' La luce ambientale è luce che non viene da nessuna direzione in particolare.
'Tutti gli oggetti nella vostra scena saranno illuminati dalla luce ambientale.
'Il secondo tipo di luce è chiamato luce diffusa.
'La luce diffusa viene creata dalla vostra fonte di luce ed è riflessa dalla superficie di un oggetto
'nella vostra scena.
```

Tutta la superficie di un oggetto che la luce colpisce direttamente sarà molto luminosa e le zone che 'la luce colpisce a mala pena saranno più scure.
' Questo crea un effetto di ombre piacevole sui lati della nostra cassa.

'La luce è creata allo stesso modo del colore. Se il primo numero è 1.0 e i due seguenti sono 0.0, 'avremo una luce rossa luminosa. Se il terzo numero è 1.0 ed i primi due sono 0.0, avremo una luce 'blu luminosa. L' ultimo numero è un valore dell' alfa. Lo lasceremo a 1.0 per ora.

'Quindi nella riga qui sotto, stiamo memorizzando i valori per una luce ambientale bianca di media 'intensità (0.5). Poiché tutti i numeri sono 0.5, avremo una luce che è a metà strada fra spenta (nero) 'e massima luminosità (bianco). Rosso, azzurro e verde mescolati nello stesso valore creeranno una 'tonalità da nero (0.0) a bianco (1.0)
' Senza una luce ambientale, i punti in cui non c'è nessuna luce diffusa appariranno molto scuri.

```
PRIVATE  lightAmbient AS Float[] = (0.5, 0.5, 0.5, 1.0)
```

'Nella riga seguente memorizziamo i valori per una luce diffusa di intensità e luminosità massime. 'Tutti i valori sono 1.0. Ciò significa che la luce è al massimo della luminosità.
'Una luce diffusa di tale luminosità illumina piacevolmente la parte anteriore della cassa.

```
PRIVATE lightDiffuse AS Float[] = (1.0, 1.0, 1.0, 1.0)
```

'Infine memorizziamo la posizione della luce.
'I primi tre numeri sono gli stessi tre numeri di gl.Translate.
' Il primo numero è per muoversi a destra e a sinistra sul piano x, il secondo numero è per muoversi 'su e giù sul piano y ed il terzo numero è per muoversi nello e fuori dallo schermo sul piano z.
'Poiché desideriamo che la nostra luce colpisca direttamente la parte anteriore della cassa, non ci 'spostiamo a sinistra o a destra in modo che il primo valore è 0.0 (nessun movimento su x), non 'desideriamo muoverci su e giù, in modo che il secondo valore è anche 0.0.
' Per il terzo valore desideriamo assicurarci che la luce sia sempre davanti la cassa. Così 'posizioneremo la luce fuori dallo schermo, verso l'osservatore. Il vetro del vostro monitor è nella 'posizione 0.0 sul piano z.
' Posizioneremo la luce in 2.0 sul piano z. Se si potesse realmente vedere la luce, starebbe sospesa in 'aria davanti al vetro dell vostro monitor.
' Facendo questo, l' unico caso in cui la luce è dietro la cassa è quando la cassa si trova anch'essa 'davanti al vetro del vostro monitor.
'Naturalmente se la cassa non fosse più dietro il vetro del vostro monitor, non potreste vederla più, 'dovunque sia la luce. Ha senso?

'Non esiste un modo facile per spiegare il terzo parametro. Si dovrebbe sapere che -2.0 si trova più 'vicino voi che -5.0 e -100.0 è LONTANO nello schermo.
' Se lo impostate a 0.0, l' immagine è troppo grande, riempie l'intero schermo.
'Non appena cominciate ad entrare nei valori positivi, l' immagine non compare più sullo schermo 'perché "è andata oltre lo schermo".
'Questo è quello che intendo quando dico fuori dallo schermo. L' oggetto è ancora là, solo che non 'potete vederlo. .

'Lasciate l' ultimo numero a 1.0. Questo indica alle OpenGL che le coordinate indicate sono la 'posizione della fonte di luce. Torneremo su questo argomento in un altro tutorial.

```
PRIVATE lightPosition AS Float[] = [0.0, 0.0, 2.0, 1.0]
```

```
 'La variabile filter qui sotto deve tenere conto di quale texture visualizzare.
'La prima texture (texture 0) è fatta usando gl_nearest (nessuno smoothing).
'La seconda texture (texture 1) utilizza il filtro gl_linear che sfuma l'immagine un bel po'.
' La terza texture (texture 2) utilizza textures mipmapped, creando una texture dall'aspetto molto
'piacevole.
'La variabile filter sarà uguale a 0, 1 o 2 secondo la texture che desideriamo usare. Cominciamo con
'la prima texture
```

texture[3] crea lo spazio in memoria per le tre texture differenti. Le textures saranno memorizzate in texture[0], texture[1] e texture[2].

```
PRIVATE filter AS Integer
PRIVATE texture AS NEW Integer[3]

'Ora carichiamo un bitmap e creiamo da esso tre texture diverse.
Per far cio andiamo nella funzione loadTextures() che abbiamo già visto nella precedente lezione

PUBLIC SUB LoadTextures()

' Questa è la sezione del codice che carica il bitmap

        logo = image.Load("pippo.png")
'  La riga qui sotto dice alle OpenGL che desideriamo sviluppare tre texture e vogliamo che le
'textures vengano memorizzate in texture[0], texture[1] ed in texture[2].

        texture = Gl.GenTextures(3)

'Nel tutorial 6, abbiamo usato texture maps con filtro linear. Richiedono una notevole quantità di
'capacità elaborativa, ma sono molto piacevoli da vedere. Il primo tipo di texture che creeremo in
'questo tutorial utilizza GL_NEAREST. Questo tipo di texture fondamentalmente non utilizza filtri.
'Richiede pochissima capacità elaborativa e non è così brutta da vedere. Se avete giocato mai a
'qualche gioco in cui le texture appaiono pixellose, probabilmente sta usando questo tipo di texture.
'L' unico beneficio di questo tipo di texture è che i progetti fatti usando questo tipo funzionano
'solitamente abbastanza bene anche sui calcolatori lenti.

'Noterete che stiamo usando GL_NEAREST sia per il MIN che per il MAG. Potete mescolare
'GL_NEAREST con GL_LINEAR e la texture apparirà un po' più piacevole, ma siamo interessati
'alla velocità, quindi useremo la qualità bassa per entrambi. Il MIN_FILTER è il filtro utilizzato
'quando un' immagine viene disegnata più piccola rispetto al formato originale della texture. Il
'MAG_FILTER è usato quando l' immagine è più grande del formato originale della texture.

        Gl.BindTexture(gl.GL_TEXTURE_2D, texture[0])
        Gl.TexImage2D(logo)
  Gl.TexParameteri(gl.GL_TEXTURE_2D, Gl.GL_TEXTURE_MIN_FILTER, gl.GL_NEAREST)
  Gl.TexParameteri(gl.GL_TEXTURE_2D, gl.GL_TEXTURE_MAG_FILTER, gl.GL_NEAREST)

'La prossima texture che costruiamo è lo stesso tipo di texture che abbiamo usato nel tutorial 6:
'Linear filtered.
'L'unica cosa che è cambiata è che memorizzeremo questa texture in texture[1] anziché in texture[0]
'perché è la nostra seconda texture.
'Se la memorizzassimo in texture[0] come sopra, scriveremmo sopra la texture di GL_NEAREST
'(la prima texture).
```

```
        Gl.BindTexture(gl.GL_TEXTURE_2D, texture[1])
        Gl.TexImage2D(logo)
  Gl.TexParameteri(gl.GL_TEXTURE_2D, Gl.GL_TEXTURE_MIN_FILTER, gl.GL_LINEAR)
  Gl.TexParameteri(gl.GL_TEXTURE_2D, Gl.GL_TEXTURE_MAG_FILTER, gl.GL_LINEAR)

'Ora un nuovo modo per fare le texture: il Mipmapping!
'Avrete notato che quando rendete un' immagine molto molto piccola sullo schermo, gran parte dei
'particolari sparisce. Patterns che all'inizio apparivano piacevoli, dopo sono brutti a vedersi.
'Quando dite alle OpenGL di costruire una texture mipmapped, OpenGL prova a costruire textures
'ad alta qualità di diverse misure.
'Quando disegnate una texture mipmapped sullo schermo, OpenGL selezionerà la texture
'MIGLIORE a vedersi, a partire da quelle che ha costruito (texture col maggior dettaglio) e la
'disegnerà sullo schermo invece di ridimensionare l'immagine originale (che causa la perdita dei
'particolari).

'Avevo detto nel tutorial 6 che c'è un modo per aggirare i limiti di 64, 128, 256, etc che le OpenGL
'hanno riguardo la larghezza e l'altezza delle textures: si tratta di glu.Build2DMipmaps.
'Da quello che ho trovato, si può usare qualsiasi immagine bitmap desiderate (di qualsiasi larghezza
'e altezza)quando si costruiscono textures mipmapped. OpenGL la ridimensionerà automaticamente
'secondo la larghezza e l'altezza adatte.

'Poiché questa è la texture numero tre, la memorizzeremo in texture[2].
'Così ora abbiamo texture[0 ] che non ha filtro, texture[1] che utilizza il linear filetring e texture[2]
'che utilizza la mipmapped texture. Per questo tutorial abbiamo finito di costruire textures.


                Gl.BindTexture(gl.GL_TEXTURE_2D, texture[2)
                Gl.TexImage2D(logo)
Gl.TexParameteri(gl.GL_TEXTURE_2D, Gl.GL_TEXTURE_MIN_FILTER,
gl.GL_LINEAR_MIPMAP_NEAREST)

  Gl.TexParameteri(gl.GL_TEXTURE_2D, gl.GL_TEXTURE_MAG_FILTER, gl.GL_LINEAR)

'La seguente riga costruisce la texture mipmapped.

                 Glu.Build2DMipmaps(logo)
END


'Ora vediamo le regolazioni di OpenGL. La prima riga di InitGL carica le texture usando il codice
'qui sopra. Dopo che le texture sono state create, abilitiamo il texture mapping 2D tramite
'gl.Enable(gl.GL_TEXTURE_2D). Lo shade mode e impostato a smooth shading.
'Il colore dello sfondo è impostato a nero; attiviamo il depth testing, quindi attiviamo il calcolo della
'prospettiva migliore che si può.

PUBLIC SUB InitGL()
        LoadTextures()
        Gl.Enable(Gl.GL_TEXTURE_2D)
        Gl.ShadeModel(gl.GL_SMOOTH)
        gl.ClearColor(0, 0, 0.0, 0.5)
        Gl.ClearDepth(1.0)
        Gl.Enable(gl.GL_DEPTH_TEST)
```

```
        Gl.DepthFunc(gl.GL_LEQUAL)
        Gl.Hint(gl.GL_PERSPECTIVE_CORRECTION_HINT, gl.GL_NICEST)


'Ora impostiamo l'illuminazione. La riga qui sotto regolerà la quantità di luce ambientale che light1
'emanerà.
'All' inizio di questo tutorial abbiamo memorizzato la quantità di luce ambientale in LightAmbient.
'Verranno utilizzati i valori che abbiamo memorizzato nell'array (luce ambientale di media
'intensità).


        gl.Lightfv(gl.GL_LIGHT1, gl.GL_AMBIENT, lightAmbient)


'Dopo imposteremo la quantità di luce diffusa che la luce numero uno emanerà.
'Abbiamo memorizzato la quantità di luce diffusa in LightDiffuse.
'Verranno usati i valori che abbiamo memorizzato in questo array (luce bianca di intensità
'massima).


        gl.Lightfv(gl.GL_LIGHT1, gl.GL_DIFFUSE, lightDiffuse)



'Ora impostiamo la posizione della luce.
'Abbiamo memorizzato la posizione in LightPosition.
'Verranno usati i valori che abbiamo memorizzato in questo array (a destra al centro della faccia
'anteriore, 0.0 sulla x, 0.0 su y e di 2 unità verso l'osservatore { fuori dallo schermo } sul piano z).



        gl.Lightfv(gl.GL_LIGHT1, gl.GL_POSITION, lightPosition)

'Infine attiviamo la luce numero uno.
'Comunque non abbiamo attivato GL_LIGHTING, in modo da non vedere ancora alcuna
'illuminazione. La luce è impostata e posizionata, è persino è attiva, ma fino a che non attiviamo
'GL_LIGHTING, la luce non funzionerà.


        gl.Enable(gl.GL_LIGHT1)
END


'Nella sezione seguente di codice, disegneremo il cubo texture mapped. Commenterò solo alcune
'linee perché sono nuove. Se non siete sicuri di cosa facciano le righe non commentate, controllare
'il tutorial 6.

PUBLIC SUB Screen_draw()

    Gl.Clear(gl.GL_COLOR_BUFFER_BIT OR gl.GL_DEPTH_BUFFER_BIT)
   Gl.LoadIdentity()
```

'Le tre righe seguenti di codice posizionano e ruotano il cubo textuyre mapped.
'gl.Translatef(0.0,0.0,z) sposta il cubo verso il valore di z sul piano z (lontano da e verso
'l'osservatore). gl.Rotatef(xrot,1.0,0.0,0.0) utilizza la variabile xrot per ruotare il cubo sull'asse x.'
'gl.Rotatef(yrot,1.0,0.0,0.0) utilizza la variabile yrot per ruotare il cubo sull'asse y.

```
Gl.Translatef(0.0, 0.0, z)
gl.Rotatef(xrot, 1.0, 0.0, 0.0)
gl.Rotatef(yrot, 0.0, 1.0, 0.0)
gl.Rotatef(0, 0, 0, 1) 'questa riga è necessaria anche se l' asse z non deve ruotare
```

'La riga seguente è simile alla riga che abbiamo usato nel tutorial 6, ma invece di impiegare
'texture[0], stiamo impiegando texture[filter]. In qualunque momento premiamo il tasto 'F', il valore
'del filtro aumenterà. Se questo valore è superiore a due, la variabile filter viene impostata di nuovo
'a zero. Quando il programma comincia filter sarà impostato a zero. Questo è come dire
'glBindTexture(GL_TEXTURE_2D, texture[0]). Se premiamo 'F' un'altra volta, variabile filter sarà
'uguale uno, che è come dire glBindTexture(GL_TEXTURE_2D, texture[1]). Usando la variabile
'filter possiamo selezionare una qualsiasi delle tre textures che abbiamo fatto.

```
gl.BindTexture(gl.GL_TEXTURE_2D, texture[filter])
```

'glNormal3f è una novità nei miei tutorial. Una normale è una linea che punta diritto dal centro di
'un poligono con un angolo di 90 gradi. Quando si usa l'illuminazione, bisogna specificare una
'normale. La normale dice alle OpenGL verso quale senso il poligono è rivestito... quale lato è in su.
'Se non specificate le normali, potrebbe accadere qualsiasi cosa di bizzarro. Le facce che non
'dovrebbero illuminarsi si illumineranno, verrà illuminato il lato sbagliato del poligono, ecc. La
'normale dovrebbe puntare verso l'esterno del poligono.

'Guardando la faccia anteriore noterete che la normale è positiva sull'asse z. Ciò significa che la
'normale punta verso l'osservatore. Esattamente la direzione in cui vogliamo che punti. Sulla faccia
'posteriore, la normale punta in direzione opposta all'osservatore, verso l'interno dello schermo.
'Ancora una volta la cosa che desideriamo. Se il cubo è ruotato di 180 gradi sulla x o sull' asse y, la
'parte anteriore sarà rivolta verso l'interno dello schermo e la parte posteriore sarà rivolta verso
'l'osservatore. Qualunque si la faccia rivolta verso l'osservatore, la normale di quella faccia punterà
'sempre verso l'osservatore. Poiché la luce è vicina all'osservatore, in qualunque momento la
'normale punterà sia verso l'osservatore che verso la luce. Quando questo accade, la faccia sarà
'illuminata. Tanto più una normale punta verso la luce, tanto più risulterà illuminata. Se entrate
'all'interno del cubo noterete che è scuro. Le normali puntano verso l'esterno, non dentro, quindi
'all'interno del cubo non c'è luce, esattamente come dovrebbe essere.

```
                // Front Face
                glNormal3f( 0.0f, 0.0f, 1.0f);                          // Normal Pointing
Towards Viewer
                glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);    // Point 1 (Front)
                glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);    // Point 2 (Front)
                glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);    // Point 3 (Front)
                glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);    // Point 4 (Front)
                // Back Face
                glNormal3f( 0.0f, 0.0f,-1.0f);                          // Normal Pointing Away
From Viewer
                glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);    // Point 1 (Back)
                glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);    // Point 2 (Back)
```

```
            glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);     // Point 3 (Back)
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);     // Point 4 (Back)
            // Top Face
            glNormal3f( 0.0f, 1.0f, 0.0f);                                 // Normal Pointing Up
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);     // Point 1 (Top)
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,  1.0f,  1.0f);     // Point 2 (Top)
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,  1.0f,  1.0f);     // Point 3 (Top)
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);     // Point 4 (Top)
            // Bottom Face
            glNormal3f( 0.0f,-1.0f, 0.0f);                                 // Normal Pointing Down
            glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);     // Point 1 (Bottom)
            glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);     // Point 2 (Bottom)
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);     // Point 3 (Bottom)
            glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);     // Point 4 (Bottom)
            // Right face
            glNormal3f( 1.0f, 0.0f, 0.0f);                                 // Normal Pointing Right
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);     // Point 1 (Right)
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);     // Point 2 (Right)
            glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);     // Point 3 (Right)
            glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);     // Point 4 (Right)
            // Left Face
            glNormal3f(-1.0f, 0.0f, 0.0f);                                 // Normal Pointing Left
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);     // Point 1 (Left)
            glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);     // Point 2 (Left)
            glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);     // Point 3 (Left)
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);     // Point 4 (Left)
        glEnd();                                                           // Done Drawing Quads
```

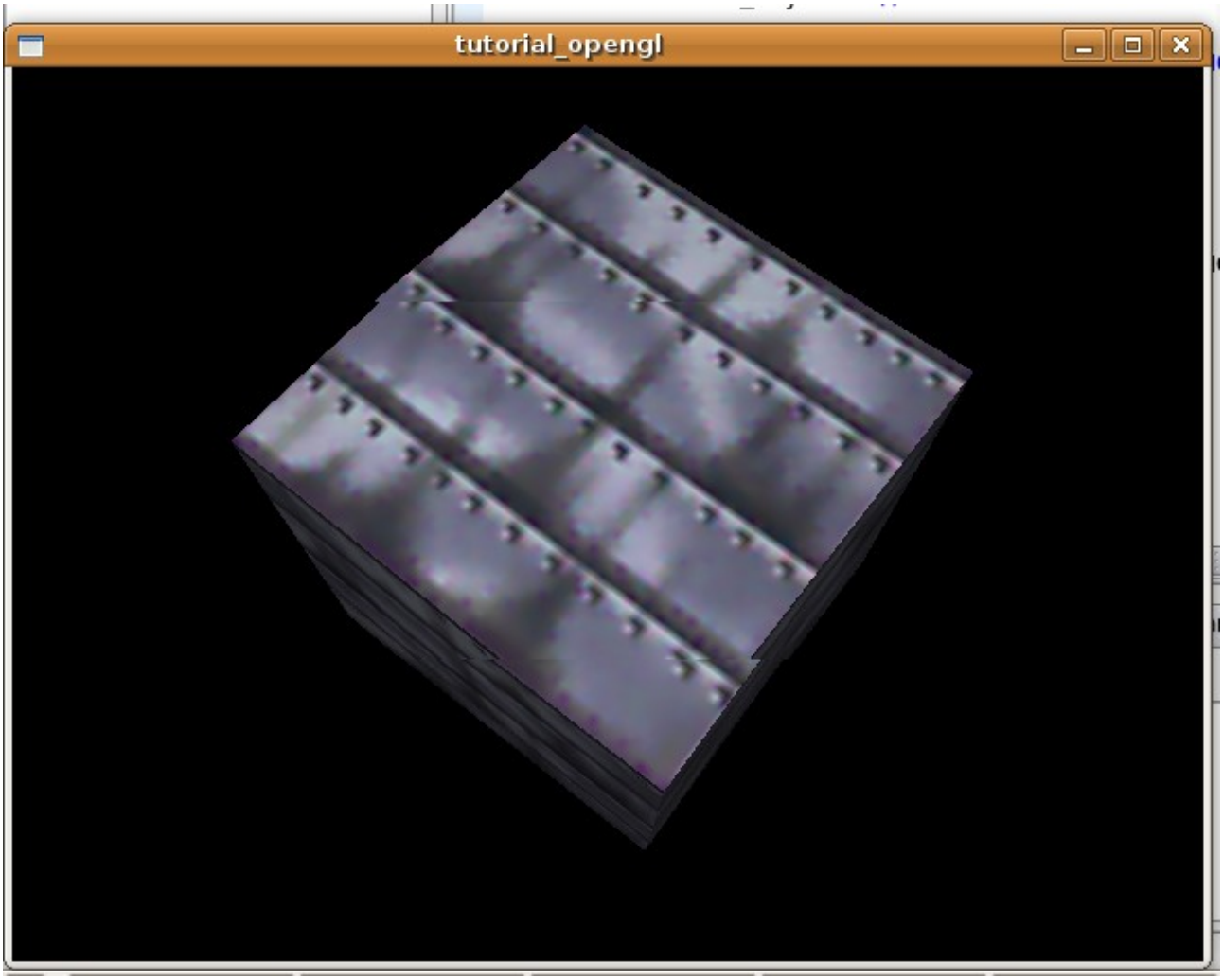
'Le due righe seguenti aumentano xrot e yrot dell'importo memorizzato in xspeed e yspeed.
'Se il valore in xspeed o yspeed è alto, xrot e yrot aumenteranno rapidamente. Più velocemente
'aumenta il valore di xrot, o di yrot, più velocemente ruoterà il cubo su quell' asse.

```
        xrot +=xspeed;
        yrot +=yspeed;

 WAIT 0.01
END
```

'Ora . Dobbiamo aggiungere il codice per accendere e spegnere illuminazione, ruotare la cassa,
'cambiare il filtro e muovere la cassa attraverso lo schermo.

```
PUBLIC SUB Screen_keyPress()

  IF key.Code = key.F1 THEN Screen.Fullscreen = NOT Screen.Fullscreen
  IF key.Code = key.Esc THEN Screen.Close()
  IF key.code = key.F2 THEN
     INC lp
     PRINT lp
     IF lp = TRUE THEN gl.Enable(gl.GL_LIGHTING)
     IF lp = FALSE THEN gl.Disable(gl.GL_LIGHTING)
```

```
  ENDIF

  IF key.Code = key.F3 THEN
     INC filter
     PRINT filter

     IF filter = 3 THEN filter = 0
  ENDIF

  IF key.code = key.Up THEN xspeed -= 0.1
  IF key.code = key.Down THEN xspeed += 0.1
  IF key.code = key.Left THEN yspeed -= 0.1
  IF key.code = key.Right THEN yspeed += 0.1
  IF key.code = key.PageUp THEN z -= 0.2
  IF key.code = key.PageDown THEN z += 0.2

END
```



Lezione 8

La maggior parte dei effetti speciali in OpenGL sono basati su un certo tipo di blending.
Il blending è usato per unire il colore di un dato pixel che sta per essere disegnato con il pixel che è già sullo schermo.
La combinazione dei colori è basato sul valore alpha dei colori e/o sulla funzione di mescolamento che è stata utilizzata. L' alfa è il quarto componente del colore, specificato solitamente per ultimo. In passato avete usato GL_RGB per specificare il colore con 3 componenti. GL_RGBA può essere usato per specificare anche l'alfa. In più, possiamo usare glColor4f() anziché glColor3f().

La maggior parte della gente pensa ad Alpha come all'indice di opacità del materiale. Un valore di Alpha di 0.0 significherebbe che il materiale è completamente trasparente. Con un valore di 1.0 sarebbe completamente opaco.

L' Equazione Di Blending
Se vi trovate a disagio con la matematica e desiderate solo vedere come fare le trasparenze, saltate questa sezione. Se desiderate capire come funziona il blending, questa sezione fa per voi.

(Rs Sr + Rd Dr, Gs Sg + Gd Dg, Bs Sb + Bd Db, As Sa + Ad Da)

OpenGL calcolerà il risultato del blending di due pixel basandosi sulla suddetta equazione. Gli indici r ed s specificano i pixel sorgenti e di destinazione. Le componenti di D ed S sono i fattori di blending. Questi valori indicano come volete sfumare i pixel. I valori più comuni per la S e la D sono (As, As, As, As) (cioè Alpha source) per la S e (1, 1, 1, 1) - (As, As, As, As) (cioè Alpha meno uno) per D. Questo produrrà un'equazione di blending simile alla seguente:

(Rs As + Rd (1 - As), Gs As + Gd (1 - As), Bs As + Bs (1 - As), As As + Ad (1 - As))

Questa equazione produrrà effetti di trasparenza/traslucenza. Blending in OpenGL.
Blending in OpenGL
Attiviamo il blending semplicemente come tutto il resto. Quindi impostiamo l'equazione e disattiviamo il depth buffer quando si disegnano oggetti trasparenti, poiché desideriamo che gli oggetti dietro la figura trasparente si vedano ancora. Questo non è il modo esatto di eseguire il blending, ma la maggior parte delle volte, nei progetti semplici funzionerà benissimo.

Rui Martins Aggiunge: Il modo corretto è di disegnare tutti i poligoni trasparenti (con Alpha < 1.0) dopo aver disegnato l' intera scena e di disegnare nell' ordine di profondità inverso (l'oggetto più lontano per primo).

Ciò è dovuto al fatto che eseguendo il blending su due poligoni (1 e 2) in ordine differente, fornisce risultati differenti, cioè (dando per assodato che il poligono 1 è più vicino all'osservatore, il modo corretto dovrebbe disegnare prima il poligono 2 e poi il poligono 1. Se lo guardate, come nella realtà, tutta la luce che arriva da dietro questi due poligoni (che sono trasparenti) deve passare prima attraverso il poligono 2, quindi attraverso il poligono 1, prima di arrivare all'occhio dell'osservatore.

Si dovrebbero ORDINARE I POLIGONI TRASPARENTI A SECONDA DELLA PROFONDITA' (DEPTH) e disegnarli DOPO CHE l' INTERA SCENA SIA STATA DISEGNATA, col DEPTH BUFFER ATTIVATO, oppure si otterrano risultati errati. So che questo a volte è faticosò, ma questo è il modo corretto di farlo.

Useremo il codice dal tutorial 7. Cominciamo aggiungendo 7 una nuova variabile nella parte

superiore del codice. Riscriverò l' intera sezione di codice per chiarezza.

```
' Gambas module file

PRIVATE CONST ScrWidth AS Integer = 640
PRIVATE CONST ScrHeight AS Integer = 480
PRIVATE Screen AS NEW Window(TRUE) AS "Screen"
PRIVATE logo AS image
PRIVATE logo1 AS Image
PRIVATE light AS Boolean
PRIVATE fp AS Boolean
PRIVATE xrot AS Float
PRIVATE yrot AS Float
PRIVATE z AS Float = -5.0
PRIVATE xspeed AS Float = 0.1
PRIVATE yspeed AS Float = 0.1
PRIVATE lightAmbient AS Float[] = [0.5, 0.5, 0.5, 1.0]
PRIVATE lightDiffuse AS Float[] = [1.0, 1.0, 1.0, 1.0]
PRIVATE lightPosition AS Float[] = [0.0, 0.0, 2.0, 1.0]
PRIVATE filter AS Integer
PRIVATE texture AS NEW Integer[3]
PRIVATE blend AS Boolean
```

Aggiungiamo le seguenti ultime due righe in qualche luogo nella sezione InitGL() del codice. Quello che questa riga fa è impostare la luminosità del disegno dell'oggetto al massimo con Alpha al 50% (opacità). Ciò significa che quando il blending è attivato, l'oggetto sarà trasparente al 50%. La seconda riga regola il tipo di blending che si sta per usare.

Rui Martins Aggiunge: Un valore di Alpha di 0.0 significherebbe che il materiale è completamente trasparente. Con un valore di 1.0 sarebbe completamente opaco.

```
        gl.Color4f(1.0, 1.0, 1.0, 0.5)
        gl.BlendFunc(gl.GL_SRC_ALPHA, gl.GL_ONE)
```

ora modificheremo la funzione Screen_keyPress() inmodo da includere la gestione del blending tramite la pressione del tasto F4

```
PUBLIC SUB Screen_keyPress()

        IF key.Code = key.F1 THEN Screen.Fullscreen = NOT Screen.Fullscreen
        IF key.Code = key.Esc THEN Screen.Close()
        IF key.code = key.F2 THEN
                INC lp
                IF lp = TRUE THEN gl.Enable(gl.GL_LIGHTING)
                IF lp = FALSE THEN gl.Disable(gl.GL_LIGHTING)
        ENDIF

        IF key.Code = key.F3 THEN
                        INC filter
                        IF filter = 3 THEN filter = 0
```

```
        ENDIF

          IF key.code = key.Up THEN xspeed -= 0.1
          IF key.code = key.Down THEN xspeed += 0.1
          IF key.code = key.Left THEN yspeed -= 0.1
          IF key.code = key.Right THEN yspeed += 0.1
          IF key.code = key.PageUp THEN z -= 0.2
          IF key.code = key.PageDown THEN z += 0.2

          IF key.code = key.F4 THEN
            INC blend
            IF blend = TRUE THEN
            gl.Enable(gl.GL_BLEND)
            gl.Disable(gl.GL_DEPTH_TEST)
          ELSE
             gl.Disable(gl.GL_BLEND)
             gl.enable(gl.GL_DEPTH_TEST)
          ENDIF
      ENDIF
END
```

Ma come possiamo specificare il colore se stiamo usando una texture map? Semplice, nel texture mode modulato, ogni pixel texturizzato è moltiplicato per il colore corrente. Così, se il colore che deve essere disegnato è (0.5, 0.6, 0.4), lo moltiplichiamo tante volte per il colore ed otteniamo (0.5, 0.6, 0.4, 0.2) (alpha si assume che sia 1.0 se non specificata).

E' tutto! Il blending è veramente piuttosto semplice da fare in OpenGL.