



UNIVERSIDAD CENTRAL DEL ECUADOR

Facultad de Ingeniería en Ciencias Físicas y Matemática

Ingeniería en Computación Gráfica

Programación Gráfica II

TEORIA DEL COLOR EN JAVA & PYTHON

Grace Pumisacho

Johana Criollo

Pamela Barrionuevo

Bryan Camacho

Fabián Portero

Javier Sigcha

Jarol Saltos

Abril 10, 2019

Resumen

El color no es más que la descomposición de la luz en millones de tonalidades y matices, que juega un papel clave en el renderizado de gráficos por computadora, puesto que es algo que se conoce y se siente a través de los ojos; por esto es necesario analizar las propiedades del color y como estas se muestran en una digitalización, claro que en esto también tiene un papel trascendental las capacidades de funcionamiento y resultado gráfico del ordenador. Los hasta 16 millones de colores que se pueden representar en un ordenador son la expansión de lo que se conoce como círculo cromático; la muestra de colores primarios, secundarios y terciarios. Los lenguajes de programación, JAVA y PHYTON, en sus entornos de renderizado cuentan con numerosos modelos de color que son usados dentro de aplicaciones gráficas a través de numerosos métodos que hacen que su implementación sea relativamente fácil, se hace uso del encapsulamiento. Este trabajo de renderizado es realizado por la librería gráfica de cada lenguaje en su entorno, misma que emplea estos métodos de manera adecuada y entendible, tanto para crear (al inicio del renderizado), para mostrar lo que se codifica en el bucle de renderizado cuando se despliega en cada fracción de tiempo lo que se dibuja en pantalla, para ajustar estos “gráficos” a una eventual redimensión en la pantalla por la que se ve la aplicación, y también para enviar todos los eventos que se desarrollan en el proceso a la tarjeta gráfica que es la que se encarga de todos los procesos de cálculos geométricos y gráficos. JAVA y PHYTON en sus librerías y métodos implementan de manera distinta la idea del color, lo que se evidencia en el manejo de distintos canales de color. Las apps realizadas a través de librerías gráficas, necesitan también mantenimiento y actualización a través del tiempo, por lo que es muy útil el modificarlas de manera segura por medio de lo que se conoce como Sistemas de Control de Versiones.

TEORIA DEL COLOR	5
¿Qué es el color?	5
Propiedades del color	5
Tono	5
Saturación	5
Luminosidad	6
¿Qué es un círculo cromático?	6
Los colores primarios	7
Los colores secundarios	7
Los colores terciarios	7
COLOR EN JAVA	8
Modelos de color	8
El cubo del color	8
Versiones más utilizadas de la función glColor	9
Colorear el fondo	9
void glColor (GLclampf red, GLclampf green, GLclampf blue,	10
GLclampf alfa)	10
void glColor (GLuint bits)	10
Colorear las primitivas	10
void glColor (GLenum modo)	11
Modelo plano (GL_FLAT)	11
Modelo suavizado (GL_SMOOTH)	11
COLOR EN PYTHON	12
pygame.Color	12
Objeto de Pygame para representaciones de color.	13
pygame.color.r	13
pygame.color.g	13
pygame.color.b	13
pygame.Color.a	13
pygame.color.cmy	13
pygame.Color.hsva	14
pygame.Color.hsla	14
pygame.Color.i1i2i3	15
pygame.Color.normalize	15
pygame.Color.correct_gamma	15
pygame.Color.set_length	16
GIT	17
Sistemas de Control de versiones (VCS)	17
¿Qué es Git?	17
Comandos de Git	17
Github	17
Git config	18
Git clone	18

Git add.....	18iv
Git rm	18
Git commit	19
Git status	19
Git branch.....	19
Git checkout	19
Git merge	20
Git reset	20
.gitignore	20
.gitkeep.....	20
Git remote	21
Git push.....	21
Git pull	21
CONCLUSIONES	22
RECOMENDACIONES	22
ANEXOS	23
EJERCICIOS JAVA	23
EJERCICIOS PYTHON	31
Lista de referencias	41

Tabla de Figuras

Fig. 1 Tono.....	5
Fig. 2 Saturación	6
Fig. 3 Luminosidad	6
Fig. 4 Circulo cromático	6
Fig. 5 Cubo del color	8
Fig. 6 Triángulo Modelo Suavizado	11

TEORIA DEL COLOR

¿Qué es el color?

El color es un atributo que percibimos de los objetos cuando hay luz. Todo el mundo que nos rodea es de colores siempre y cuando esté iluminado.

La Teoría del Color es un conjunto de conocimientos y normas que permiten manejar los colores, sean de luz o pigmentos, para conseguir el efecto deseado. (Estrada, 2018)

Los colores también despiertan respuestas emocionales específicas en las personas. La **gama cromática fría** a la que pertenece el azul y sus derivados son relajantes, tranquilizantes, expresan soledad y lejanía.

Por otro lado, la **gama cálida** a la que pertenecen el amarillo, el rojo y sus derivados son colores excitantes, expresan dinamismos, proximidad, fuerza, alegría y evocan al fuego y al sol. (Bender, 2019)

Propiedades del color

Las propiedades son aquellos atributos que cambian y hacen único a cada color.

Tono

Es conocido como matiz, tinte, croma o por su nombre en inglés, Hue. Es la propiedad que diferencia un color de otro y por la cual designamos los nombres de los colores: verde, violeta, rojo, etc. Hace referencia al recorrido que hace un color en el círculo cromático adquiriendo matices, como por ejemplo el rojo anaranjado o el amarillo verdoso. (Estrada, 2018)



Fig. 1 Tono

Saturación

Representa la intensidad cromática o pureza de un color. En otras palabras, es la claridad u oscuridad de un color y está determinada por la cantidad de luz (o mezcla de blanco) que un color tiene. (Estrada, 2018)



Fig. 2 Saturación

Luminosidad

Es la cantidad de luz que refleja una superficie en comparación con la reflejada por una superficie blanca en iguales condiciones de iluminación. Más luminosidad la acerca al blanco, menos al negro. (Estrada, 2018)



Fig. 3 Luminosidad

¿Qué es un círculo cromático?

El círculo cromático, también llamado a veces **rueda de colores** no es más que una representación gráfica de los colores en que se descompone la luz natural.

Puede ser un círculo que represente sólo los colores primarios, o pueden sumársele los secundarios y los terciarios. (Bender, 2019)



Fig. 4 Círculo cromático

Los colores primarios

El origen de los millones de colores que podemos ver son los que conocemos como Colores Primarios, o primitivos. Estos son los que no pueden conseguirse a través de la suma o mezcla de otros colores. En realidad, es un modelo ideal basado en la percepción del ojo humano, pero nos sirve para entendernos. (Bender, 2019)

Los colores secundarios

Son los que se forman con la mezcla de dos colores primarios. Según el modelo clásico RYB son el Violeta, el Verde y el Naranja. (Bender, 2019)

Los colores terciarios

Son aquellos que surgen de la combinación de un color primario con otro secundario. Esto da lugar a distintas variedades de tonos que van desde el rojo violáceo, el rojo anaranjado, al amarillo anaranjado, pasando por el amarillo verdoso, el azul verdoso y el azul violáceo. (Bender, 2019)

COLOR EN JAVA

Modelos de color

OpenGL incorpora dos modelos de color: modo RGBA y modo índices de color. En el primer caso el color se define a partir de tres valores de color (correspondientes al rojo, al verde y al azul) y, opcionalmente, un valor de alfa, que indica la translucidez del color. En el modo de índices de color, los colores se encuentran ordenados y se accede a ellos a través de un único valor que se corresponde con un índice asociado al color. El modo RGBA es el más versátil y el más utilizado. El modo de índices de color se utiliza en casos muy específicos.

El cubo del color

Los colores en modo se pueden representar utilizando un cubo de lado 1, donde cada eje representa uno de los tres colores. Los demás vértices representan los colores amarillo, magenta, cian, blanco y negro. Cada punto en el interior del cubo representa un color, y sus coordenadas indican la cantidad de rojo, verde y azul que lo componen. Sobre la línea que une dos colores (dos puntos) se encuentra toda la graduación existente entre los dos. Así entre el blanco y el negro, se representa toda la gama de grises.

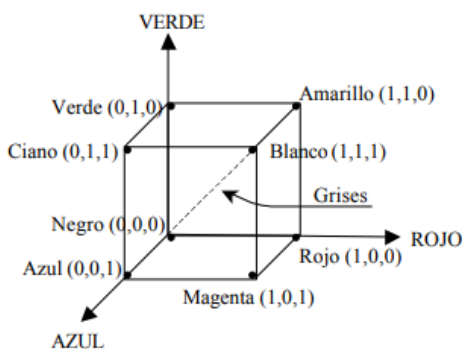


Fig. 5 Cubo del color

Para seleccionar un color debemos utilizar la función `glColor`, cuyos parámetros son los valores RGB del color. Una vez seleccionado un color, todo lo que se dibuje se coloreará con ese color, hasta que cambiemos a otro. La función `glColor` tiene varias versiones, según se utilice la componente alfa de translucidez, y según el tipo de los argumentos. Con respecto al tipo, podemos utilizar dos notaciones: con tipos reales, los

parámetros toman valores entre 0 (ausencia del color) hasta 1 (saturación total del color), tal y como hemos visto en la representación con el cubo. Con tipos enteros, los valores se toman entre 0 (ausencia del color) hasta 255 (saturación total del color), al estilo de cómo lo hacen otros sistemas.

Versiones más utilizadas de la función glColor

- void glColor3f (GLfloat red, GLfloat green, GLfloat blue)
- void glColor4f (GLfloat red, GLfloat green, GLfloat blue, GLfloat alfa)
- void glColor3i (GLint red, GLint green, GLint blue)
- void glColor4i (GLint red, GLint green, GLint blue, GLint alfa)

Red, green, blue: Valores para el rojo, verde y azul. En las versiones de tipo real toman valores en el intervalo [0, 1]; en las de tipo entero en [0, 255].

Alfa: Valor para la componente de translucidez. En las versiones de tipo real toma valores en el intervalo [0, 1]; en las de tipo entero en [0, 255].

La cantidad de colores que nuestro ordenador es capaz de representar depende de la tarjeta de vídeo de que dispongamos. Así, si la tarjeta tiene una profundidad de color (la profundidad es el número de bits que utilizamos para representar el color de cada pixel) de 4 bits, puede representar 16 colores; si la profundidad es de 8 bits, el número de colores disponibles es de 256; y si es de 24 bits, puede representar más de 16 millones de colores. Si al utilizar la función glColor seleccionamos un color del que no dispone nuestro sistema, OpenGL se encarga de seleccionar en su lugar el color más cercano al elegido.

Colorear el fondo

Al igual que el buffer de profundidad o Z-buffer, OpenGL mantiene un buffer de color que indica el color en que debe dibujarse cada pixel. Para borrar el área de dibujo y colorearla a un color, basta con inicializar ese buffer al color deseado. Para ello utilizamos las funciones glClearColor y glClear (esta última función es la misma que permite inicializar el buffer de profundidad).

void glColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alfa)

Red, green, blue, alfa: son las cuatro componentes del color. El tipo GLclampf es un tipo real, por lo tanto estos argumentos toman valores en el intervalo [0, 1].

Esta función indica el color con que debe inicializarse el buffer de color.

void glClear (GLuint bits)

Bits: indica el elemento que se inicializa. Para inicializar el buffer de color debemos utilizar el valor GL_COLOR_BUFFER_BIT.

Puesto que muy habitualmente tanto el buffer de color como el de profundidad deben ser inicializados cada vez que se redibuja la escena, puede llamarse una sola vez a la función glClear utilizando un conector de tipo OR:

`glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`

Colorear las primitivas

La función glColor que ya hemos visto sirve para seleccionar el color en que se dibujan todos los elementos que se diseñen a continuación. Cuando definimos una primitiva de diseño, el color se asigna a cada uno de sus vértices y no a la primitiva completa. Esto implica que puede haber primitivas como líneas o polígonos en las que cada vértice se dibuje de un color. Entonces ¿cómo se colorea el interior de la primitiva?

En el caso de los puntos no hay problema. Puesto que el vértice tiene asignado un color, el punto se dibujará con ese color. El caso de las líneas y los vértices es más complicado. El coloreado del interior depende, en estos casos, del modelo de sombreado (shading model) elegido: plano o suavizado. El modelo de sombreado se elige con la función glShadeModel que afecta a todas las primitivas que se definan posteriormente, hasta que sea alterado.

void glShadeModel (GLenum modo)

modo: modelo de sombreado elegido. Puede valer GL_FLAT (modelo plano) o GL_SMOOTH (modelo suavizado, que es el valor por defecto).

Modelo plano (GL_FLAT)

El interior de la línea o del polígono se colorea completamente a un único color. Este color es el correspondiente al último vértice de la primitiva, en el caso de las líneas, o al primer vértice de la primitiva, en el caso de los polígonos.

Modelo suavizado (GL_SMOOTH)

El interior de las primitivas se colorea haciendo una interpolación o suavizado entre los colores asociados a los vértices. Así, si una línea tiene un vértice de color amarillo y otro de color azul, la parte interior se colorea en toda la gama de colores que van del amarillo al azul. Lo mismo ocurre en el caso de los polígonos, aunque se juega en este caso con tres, cuatro o más colores. El degradado puede representarse sobre el cubo de color mediante la recta que une dos colores, en el caso de las rectas, o el plano que une tres colores, en el caso de los triángulos, etc.

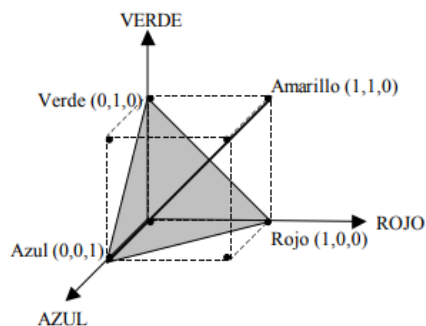


Fig. 6 Triángulo Modelo Suavizado

Evidentemente, el modelo de sombreado elegido sólo afecta cuando los vértices de las primitivas tienen colores distintos. Si los colores fueran iguales, la representación equivale al modelo plano. (Molina Carmona & Puchol García, 1999)

COLOR EN PYTHON

Python es bastante capaz de ejecutar juegos. Es probable que incluso te sorprenda cuánto es posible en menos de 30 milisegundos. Aun así, no es difícil alcanzar el techo una vez que tu juego comienza a volverse más complejo. Cualquier juego que se ejecute en tiempo real hará uso completo de la computadora. (Shinners, 2000)

pygame.Color

La clase de `RGBaColor` representa los valores de color utilizando un rango de valores de 0-255. Permite operaciones aritméticas básicas - operaciones binarias `+`, `-`, `*`, `//`, `%`, y la operación singular `~` - para crear nuevos colores, soporta conversiones a otros espacios de color como HSV o HSL y le permite ajustar los canales de color individuales. Alfa por defecto es 255 cuando no se da. Las operaciones aritméticas y el `correct_gamma()` método conservan las subclases. Para los operadores binarios, la clase del color devuelto es la del objeto de color de la mano izquierda del operador.

'`rgbvalue`' puede ser un nombre de color, una HTML cadena de formato de color, una cadena de número hexadecimal o un valor de píxel entero. El HTML formato es '`#rrggbbaa`', donde `rr`, `gg`, `bb` y `aa` son números hexadecimales de 2 caracteres. El alfa `aa` es opcional. Una cadena de número hexadecimal tiene la forma '`0xrrggbbaa`', donde `aa` es opcional.

Los objetos de color admiten la comparación de igualdad con otros objetos de color y tuplas de enteros de 3 o 4 elementos.

Los objetos de color exportan la interfaz de matriz de nivel C. La interfaz exporta una matriz de bytes sin signo unidimensional de solo lectura de la misma longitud asignada que el color. Para CPython 2.6 y versiones posteriores, la nueva interfaz de búfer también se exporta, con las mismas características que la interfaz de matriz.

Los operadores de división de piso `//` y módulo `%` no generan una excepción para la división por cero. En cambio, si un canal de color, o alfa, en el color de la mano derecha es 0, el resultado es 0. Por ejemplo:

```
# These expressions are True
Color(255, 255, 255, 255) // Color(0, 64, 64, 64) == Color(0, 3, 3, 3)
Color(255, 255, 255, 255) % Color(64, 64, 64, 0) == Color(63, 63, 63, 0)
```

Objeto de Pygame para representaciones de color.

Color(name) -> Color

Color(r, g, b, a) -> Color

Color(rgbvalue) -> Color

La nueva implementación de Color se realizó en pygame 1.8.1.

pygame.color.r

Obtiene o establece el valor rojo del Color.

r -> int

pygame.color.g

Obtiene o establece el valor verde del Color.

g -> int

pygame.color.b

Obtiene o establece el valor azul del Color.

b -> int

pygame.Color.a

Obtiene o establece el valor alfa del Color.

a -> int

pygame.color.cmy

Obtiene o establece la representación CMY del Color.

cmy -> tuple

Los CMY componentes están en los rangos C= [0, 1], M= [0, 1], Y= [0, 1]. Tenga en cuenta que esto no devolverá los CMY valores absolutamente exactos para los RGB valores establecidos en todos los casos. Debido a la RGB asignación de 0-255 y la CMY asignación de errores de redondeo de 0-1 puede hacer que los CMY valores difieran ligeramente de lo que podría esperar.

pygame.Color.hsva

Obtiene o establece la representación HSVA del Color.

hsva -> tuple

Los HSVA componentes están en los rangos H= [0, 360], S= [0, 100], V= [0, 100], A = [0, 100]. Tenga en cuenta que esto no devolverá los HSV valores absolutamente exactos para los RGB valores establecidos en todos los casos. Debido a la RGB asignación de 0-255 y la HSV asignación de errores de redondeo de 0-100 y 0-360 puede hacer que los HSV valores difieran ligeramente de lo que podría esperar.

pygame.Color.hsla

Obtiene o establece la representación HSLA del Color.

hsla -> tuple

La HSLA representación del color. Los HSLA componentes están en los rangos H= [0, 360], S= [0, 100], V= [0, 100], A = [0, 100]. Tenga en cuenta que esto no devolverá los HSL valores absolutamente exactos para los RGB valores establecidos en todos los casos. Debido a la RGB asignación de 0-255 y la HSL

asignación de errores de redondeo de 0-100 y 0-360 puede hacer que los HSL valores difieran ligeramente de lo que podría esperar.

pygame.Color.i1i2i3

Obtiene o establece la representación I1I2I3 del Color.

i1i2i3 -> tuple

La I1I2I3representación del color. Los I1I2I3componentes están en los rangos I1= [0, 1], I2= [-0.5, 0.5], I3= [-0.5, 0.5]. Tenga en cuenta que esto no devolverá los I1I2I3 valores absolutamente exactos para los RGB valores establecidos en todos los casos. Debido a la RGB asignación de 0-255 y la I1I2I3asignación de errores de redondeo de 0-1 puede hacer que los I1I2I3valores difieran ligeramente de lo que podría esperar.

pygame.Color.normalize

Devuelve los valores RGBA normalizados del Color.

normalize () -> tuple

Devuelve los RGBA valores normalizados del Color como valores de punto flotante.

pygame.Color.correct_gamma

Aplica un cierto valor gamma al Color.

correct_gamma (gamma) -> Color

Aplica un cierto valor de gamma al Color y devuelve un nuevo Color con los RGBA valores ajustados.

pygame.Color.set_length

Establezca el número de elementos en el Color en 1,2,3, o 4.

set_length (len) -> None

La longitud de color predeterminada es 4. Los colores pueden tener longitudes 1,2,3 o 4. Esto es útil si desea desempaquetar en r, g, b y no r, g, b, a. Si quieres obtener la longitud de un color hazlo len(acolor). (Shinners, pygame, 2000)

GIT

Sistemas de Control de versiones (VCS)

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. No solo el código fuente puede estar como archivos bajo control de versiones, en realidad cualquier tipo de archivo que encuentres en un ordenador puede ponerse bajo control de versiones.

Por ejemplo, si un diseñador gráfico o web, y quieres mantener cada versión de una imagen o diseño (algo que sin duda quieres), un sistema de control de versiones (Version Control System o VCS en inglés) es una elección muy sabia. Te permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más. Usar un VCS también significa generalmente que, si fastidias o pierdes archivos, puedes recuperar.

¿Qué es Git?

Es un software de control de versiones distribuido diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Git es uno de los sistemas de control de versiones más utilizado hoy en día por su rapidez y la gran cantidad de funcionalidades que ofrece. Es posible también que el éxito de Github esté propiciando la paulatina adopción de git en detrimento de otros sistemas como subversionarlos fácilmente. (Cloud Computing, 2019)

Además, obtienes todos estos beneficios a un coste muy bajo. (Git, 2019)

Comandos de Git

Github

Es una forja para alojar proyectos utilizando el sistema de control de versiones Git. Github permite alojar de forma gratuita un número indeterminado de repositorios públicos y es ampliamente utilizado hoy en día por grandes proyectos de software, así como por proyectos desarrollados por un solo individuo. (Cloud Computing, 2019)

Git config

Es un comando de que se utiliza una vez instalado git que nos permite obtener y establecer variables de configuración que controlan todos los aspectos del funcionamiento de Git. Las variables de configuración de Git se pueden almacenar en tres lugares diferentes:

Git clone

Es un comando que se puede utilizar crea al momento de instalar git que permite clonar un repositorio en un directorio recién creado, crea sucursales de seguimiento remoto para cada sucursal en el repositorio clonado. (Atlassian Bitbucket, 2018)

Git add

El git add comando agrega un cambio en el directorio de trabajo al área de preparación. Le dice a Git que desea incluir actualizaciones de un archivo en particular en la próxima confirmación. Sin embargo, git add realmente no afecta al repositorio de ninguna manera significativa: los cambios no se registran realmente hasta que se ejecuta `git commit`.

Junto con estos comandos, también `git status` deberá ver el estado del directorio de trabajo y el área de preparación. (Atlassian Bitbucket, 2018)

Git rm

Es un comando que se puede utilizar para eliminar archivos individuales o una colección de archivos. La función principal de `git rm` es eliminar los archivos rastreados del índice Git. Además, `git rm` se puede usar para eliminar archivos tanto del índice de preparación como del directorio de trabajo. No hay opción para eliminar un archivo solo del directorio de trabajo. Los archivos en los que se está operando deben ser idénticos a los archivos en la versión actual HEAD. Si hay

una discrepancia entre la HEAD versión de un archivo y el índice de preparación o la versión del árbol de trabajo, Git bloqueará la eliminación. Este bloque es un mecanismo de seguridad para evitar la eliminación de cambios en curso. (Atlassian Bitbucket, 2018)

Git commit

El git commit comando captura una instantánea de los cambios organizados actualmente del proyecto. Las instantáneas comprometidas pueden considerarse versiones “seguras” de un proyecto: Git nunca las cambiará a menos que usted lo solicite explícitamente. Antes de la ejecución de git commit, el git add comando se utiliza para promover o "modificar" los cambios en el proyecto que se almacenarán en una confirmación. Estos dos comandos git commit y git add son dos de los más utilizados (Atlassian Bitbucket, 2018)

Git status

El git status comando muestra el estado del directorio de trabajo y el área de preparación. Le permite ver qué cambios se han programado, cuáles no, y qué archivos no están siendo rastreados por Git. La salida de estado no muestra ninguna información sobre el historial de proyectos confirmado. Para esto, necesitas usar git log. (Atlassian Bitbucket, 2018)

Git branch

El git branch comando le permite crear, listar, renombrar y eliminar ramas. No te permite cambiar entre ramas o volver a unir el historial bifurcado. Por esta razón, git branch está estrechamente integrado con los comandos git checkout y git merge.

El comando git branch se utiliza para ver y visitar otras ramas. Al invocar el comando, git branch -ase devolverá una lista de todos los nombres de rama conocidos. (Atlassian Bitbucket, 2018)

Git checkout

El git checkout comando opera sobre tres entidades distintas: archivos, confirmaciones y ramas. Además de la definición de "verificación", la frase "verificación" se usa comúnmente para implicar el acto de ejecutar el git checkout

comando. En el tema Deshacer cambios , vimos cómo git checkout se puede usar para ver confirmaciones antiguas.

El git checkout comando te permite navegar entre las ramas creadas por git branch. (Atlassian Bitbucket, 2018)

Git merge

El git merge comando le permite tomar las líneas de desarrollo independientes creadas por git branch integrarlas en una sola rama.

Git merge combinará múltiples secuencias de confirmaciones en una historia unificada. En los casos de uso más frecuentes, git merge se utiliza para combinar dos ramas. (Atlassian Bitbucket, 2018)

Git reset

El git reset comando es una herramienta compleja y versátil para deshacer cambios. Tiene tres formas primarias de invocación. Estas formas corresponden a los argumentos de la línea de comando --soft, --mixed, --hard. Cada uno de los tres argumentos corresponde a los tres mecanismos internos de gestión de estado de Git, El árbol de compromisos (HEAD), El Índice de estadificación y El Directorio de trabajo. (Atlassian Bitbucket, 2018)

.gitignore

Git tiene una herramienta imprescindible casi en cualquier proyecto, el archivo ".gitignore", que sirve para decirle a Git qué archivos o directorios completos debe ignorar y no subir al repositorio de código.

Su implementación es muy sencilla, por lo que no hay motivo para no usarlo en cualquier proyecto y para cualquier nivel de conocimientos de Git que tenga el desarrollador. Únicamente se necesita crear un archivo especificando qué elementos se deben ignorar y, a partir de entonces, realizar el resto del proceso para trabajo con Git de manera habitual. (Desarrollo Web, 2016)

.gitkeep

La limitación de Git de no gestionar carpetas vacías y poder solucionar el problema de compartir estructuras de carpetas iniciales, una convención que se suele utilizar por muchas personas es la de crear dentro de cualquier carpeta vacía un archivo vacío llamado gitkeep.

Estos archivos no ocupan nada, ya que están vacíos, por lo que no añaden apenas nada tampoco a la carga del remoto de Git ni a la transferencia de datos. Es como si no estuvieran. Pero como están Git los registra en el control de código y por tanto añade también la carpeta que los contiene.

Git remote

El git remote comando le permite crear, ver y eliminar conexiones a otros repositorios. Las conexiones remotas son más como marcadores en lugar de enlaces directos a otros repositorios. En lugar de proporcionar acceso en tiempo real a otro repositorio, sirven como nombres convenientes que pueden usarse para hacer referencia a una URL no tan conveniente (Atlassian Bitbucket, 2018)

Git push

El git push comando se utiliza para cargar contenido del repositorio local en un repositorio remoto. Presionar es la forma en que transfiere las confirmaciones desde su repositorio local a un repositorio remoto. Es la contraparte de git fetch, pero mientras que la obtención de importaciones se compromete con las sucursales locales, las exportaciones se comprometen a sucursales remotas. Las ramas remotas se configuran usando el git remote comando. Empujar tiene el potencial de sobrescribir los cambios, se debe tener cuidado al empujar. (Atlassian Bitbucket, 2018)

Git pull

El git pull es un comando que se usa para obtener y descargar contenido de un repositorio remoto y actualizar inmediatamente el repositorio local para que coincida con ese contenido. La combinación de cambios remotos en el repositorio local es una tarea común en los flujos de trabajo de colaboración basados en Git. El git pull comando es en realidad una combinación de otros dos comandos, git fetch seguido por git merge. (Atlassian Bitbucket, 2018)

CONCLUSIONES

- Tanto las librerías JOGL como PHY GAME dentro de los lenguajes de programación JAVA y PHYTON respectivamente son potentes herramientas para la representación gráfica a partir de primitivas (polígonos, triángulos, círculos, etc.) permitiendo así brindar una más amplia gama de posibilidades dentro del desarrollo de aplicaciones afines, con un manejo muy versátil del color y sus diferentes aplicaciones.
- Es posible encontrar varias similitudes y diferencias entre ambas librerías (de tipo sintáctico comúnmente), pero cabe destacar que es posible obtener los mismos resultados concernientes al tema propuesto y en muchos más.
- Git sin lugar a duda puede presentarse de forma indispensable en el proceso de creación y manejo de proyectos a pequeña y gran escala, ofreciendo una gran variedad de recursos administrativos para los mismos permitiendo un desarrollo más óptimo de los proyectos en mención.

RECOMENDACIONES

- El estudio de los colores es muy importante ya que cada lenguaje maneja el sistema hexadecimal es decir colores primarios RGB siendo los colores rojo, verde y azul los fundamentales para crear nuevos colores.
- En Python, además de opengl se necesita una librería llamada Pygame la cual su función principal son juegos en 2D, pero ayuda a OpenGL y el manejo de los colores en RGB.
- Para nuestro estudio de Ingeniería en Computación Grafica es muy importante los colores, ya que, al desarrollo de una aplicación, un juego, etc. Los colores influyen y cada color tiene su significado.

ANEXOS

EJERCICIOS JAVA

Para los ejemplos se presentarán ejercicios con el código e imágenes del resultado, en los ejercicios tenemos clases en común que son:

```
package org.yourorghere;
import com.sun.opengl.util.Animator;
import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.media.opengl.GL;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCanvas;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.glu.GLU;

public class EJEMPLO implements GLEventListener

    float rx=0, rx1=0;
    public static void main(String[] args) {

        Frame frame = new Frame("Simple JOGL Application");
        GLCanvas canvas = new GLCanvas();
        canvas.addGLEventListener(new EJEMPLO());
        frame.add(canvas);
        frame.setSize(640, 480);
        final Animator animator = new Animator(canvas);
        frame.addWindowListener(new WindowAdapter() {

            public void windowClosing(WindowEvent e) {
                new Thread(new Runnable() {
                    public void run() {
                        animator.stop();
                        System.exit(0);
                    }
                }).start();
            }
        });
        // Center frame
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
        animator.start();
    }

    public void init(GLAutoDrawable drawable) {
        GL gl = drawable.getGL();
        System.err.println("INIT GL IS: " + gl.getClass().getName());
        gl.setSwapInterval(1);
    }
}
```

```

gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
gl.glShadeModel(GL.GL_SMOOTH);
gl.glEnable(GL.GL_DEPTH_TEST);
gl.glEnable(GL.GL_BLEND);

gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE_MINUS_SRC_ALPHA);
}

public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
height)
{

    GL gl = drawable.getGL();
    GLU glu = new GLU();
    if (height <= 0) { // avoid a divide by zero error!
        height = 1;
    }
    final float h = (float) width / (float) height;
    gl.glViewport(0, 0, width, height);
    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();
    glu.gluPerspective(45.0f, h, 1.0, 20.0);
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
}

```

En el primer ejercicio se muestra la diferencia entre la forma en la que se construyen figuras básicas, como la diferencia entre “Triangle Fan” y “Triangle Strip” la cuál nos indica la manera en que el envolvente del triángulo cerrará los vértices establecidos.

Esta diferencia también se la encuentra en las primitivas “Lines” teniendo “Line Fan” y “Line Strip”.

Todo esto podemos apreciar siempre y cuando el sombreador “ShadeModel” esté en Smooth, la cual se activa con la siguiente línea de código: “gl.glShadeModel(GL.GL_SMOOTH);”.

Es necesario conocer cómo rellenar de color a cada cara de un plano, los cuales se pueden acceder solamente con GL_POLYGON, teniendo gl.glPolygonMode(GL.GL_FRONT, GL.GL_FILL) y gl.glPolygonMode(GL.GL_BACK, GL.GL_LINE); implementándolo de la siguiente manera:

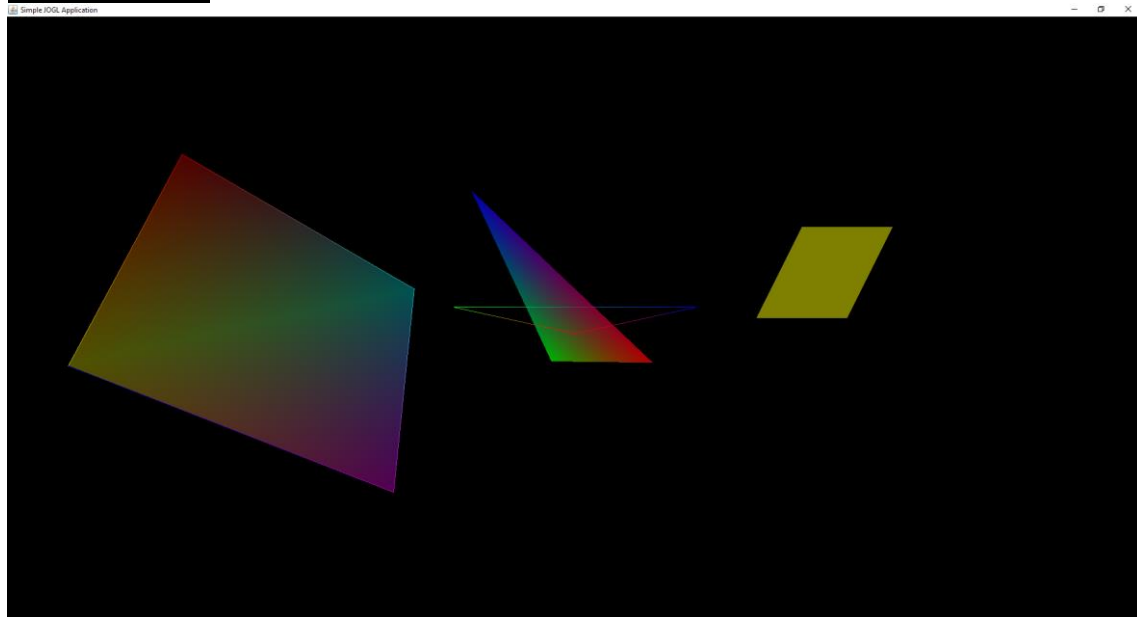
```

gl.glPolygonMode(GL.GL_BACK, GL.GL_LINE);
gl.glBegin(gl.GL_POLYGON);
//gl.glPolygonMode(GL.GL_FRONT, GL.GL_FILL);
gl.glColor3f(0,.3f,.3f);
gl.glVertex2d(117, 87);
gl.glVertex2d(117, 10);
gl.glVertex2d(190, 10);
gl.glVertex2d(190, 87);
gl.glFlush();
gl.glEnd();

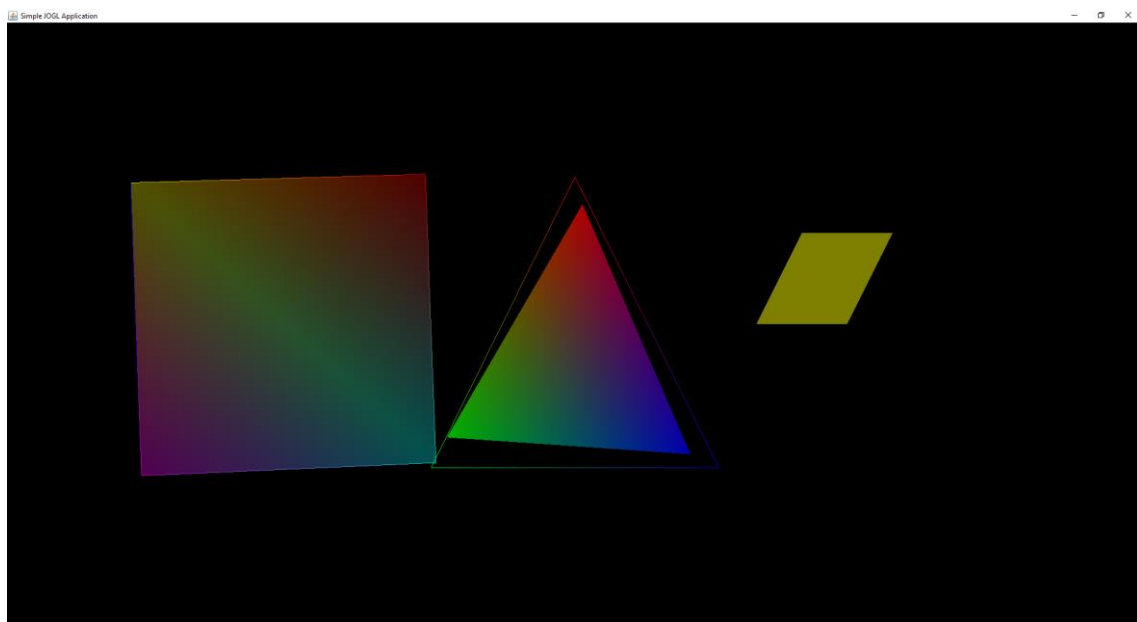
```

A continuación, presentaremos los ejercicios propuestos así como su clase display donde se encuentran sus vértices y los colores correspondientes:

EJERCICIO 1



1



2

Podemos apreciar en la ejecución del programa la interpolación de colores cuando el sombreador “ShadeModel” está activo, teniendo cada uno de los vértices un color distinto.

A continuación, se presentará el código del Ejercicio1:

¹ EJERCICIO 1

² EJERCICIO 1

```
public void display(GLAutoDrawable
drawable) {
```

```
    gl.glClear(GL.GL_COLOR_BUFFER_
    BIT | GL.GL_DEPTH_BUFFER_BIT);
    gl.glLoadIdentity();
    gl.glTranslatef(0f, 0.0f, -5.0f);
    gl.glRotatef(rx,1,0,0);
    gl.glBegin(GL.GL_LINE_LOOP);
        gl.glColor3f(1.0f, 0.0f, 0.0f);
        gl.glVertex2f(0.0f, 1.0f);
        gl.glColor3f(0.0f, 1.0f, 0.0f);
        gl.glVertex2f(-1.0f, -1.0f);
        gl.glColor3f(0.0f, 0.0f, 1.0f);
        gl.glVertex2f(1.0f, -1.0f);
    gl.glEnd();

    gl.glLoadIdentity();

    gl.glTranslatef(0f, 0.0f, -6.0f);
    gl.glRotatef(rx 1,1,1f,1);

    gl.glBegin(GL.GL_TRIANGLES);
        gl.glColor4f(1.0f, 0.0f, 0.0f,.7f);
        gl.glVertex2f(0.0f, 1.0f); // Top
        gl.glColor4f(0.0f, 1.0f, 0.0f,.7f);
        gl.glVertex2f(-1.0f, -1.0f);
        gl.glColor4f(0.0f, 0.0f, 1.0f,.7f);
        gl.glVertex2f(1.0f, -1.0f);
    gl.glEnd();

    gl.glLoadIdentity();

    gl.glTranslatef(-2f, 0.0f, -5.0f);
    gl.glRotatef(rx,0,1f,1);

    gl.glBegin(GL.GL_LINE_STRIP);
        gl.glColor3f(.8f, 0.8f, 0f);
        gl.glVertex2f(-1.0f, 1.0f);
        gl.glColor3f(.8f, 0f, 0f);
        gl.glVertex2f(1.0f, 1.0f);
        gl.glColor3f(0.0f, 0.8f, 0.8f);
        gl.glVertex2f(1.0f, -1.0f);
        gl.glColor3f(0.8f, 0.0f, 0.8f);

        gl.glVertex2f(-1.0f, -1.0f);
        gl.glColor4f(0f, 0.0f, 1f,.8f);
        gl.glVertex2f(-1.0f, 1.0f);
    gl.glEnd();
```

```
GL gl = drawable.getGL();
```

```
    gl.glBegin(GL.GL_QUADS);
        gl.glColor4f(.8f, 0.8f, 0f,.4f);

        gl.glVertex2f(-1.0f, 1.0f); // Top
        gl.glColor4f(.8f, 0f, 0f,.4f); //
    Set the current drawing color to red
        gl.glVertex2f(1.0f, 1.0f);
        gl.glColor4f(0.0f, 0.8f, 0.8f,.4f);
    // Set the current drawing color to green
        gl.glVertex2f(1.0f, -1.0f); //
    Bottom Left
        gl.glColor4f(0.8f, 0.0f, 0.8f,.4f);
    // Set the current drawing color to blue
        gl.glVertex2f(-1.0f, -1.0f); //
    Bottom Right
        gl.glColor4f(0f, 0.0f, 1f,1f);

        gl.glVertex2f(-1.0f, 1.0f);
        gl.glEnd();

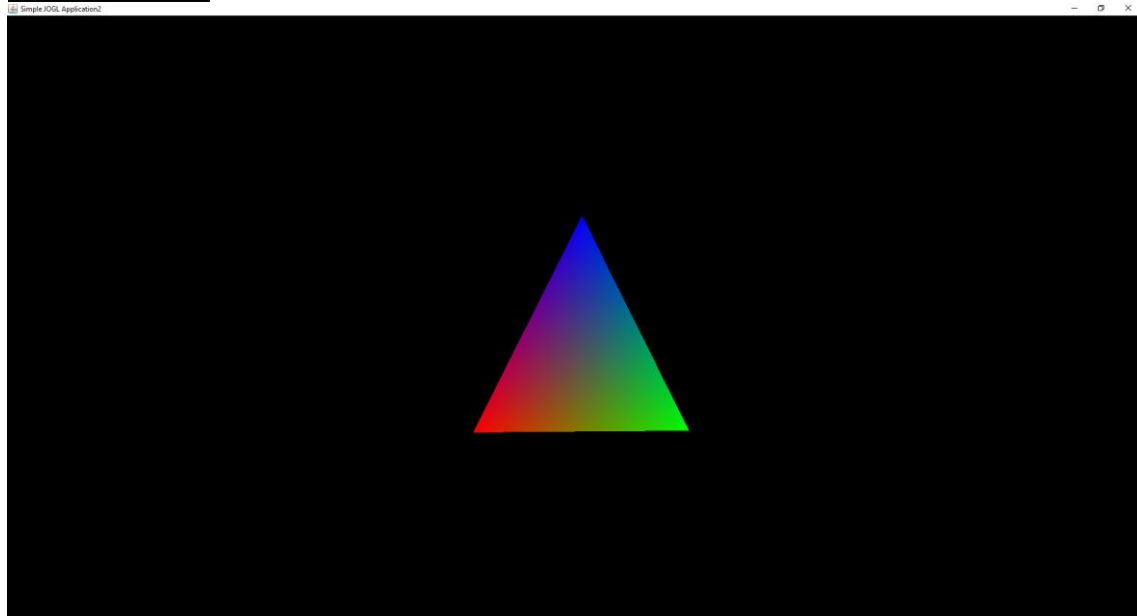
    gl.glLoadIdentity();
    gl.glTranslatef(1f, 0.0f, -5f);
    gl.glRotatef(rx,0,1,0);
    gl.glLoadIdentity();
    gl.glTranslatef(1f, 0.0f, -4.0f);

    gl.glBegin
    (gl.GL_TRIANGLE_STRIP);
        gl.glColor4f(1,1,0,.5F);
        gl.glVertex3f (0, 0, 0);
        gl.glVertex3f (.50f, 0, 0);
        gl.glVertex3f (.25f, .50f, 0);
        gl.glVertex3f (.75f, .50f, 0);
    gl.glEnd ();

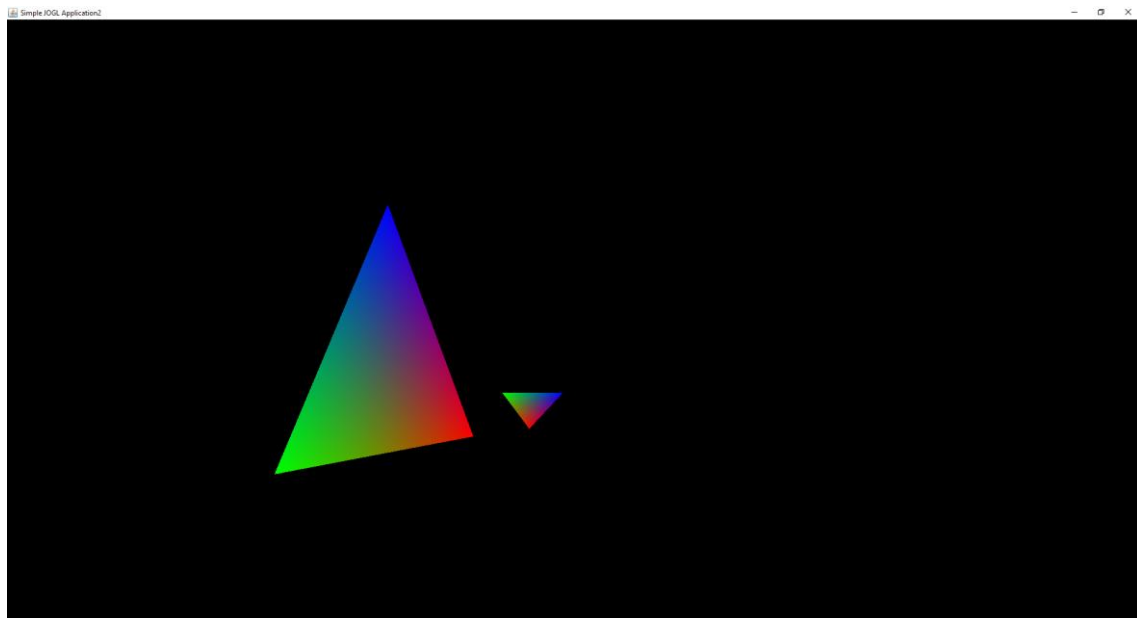
    rx+=1f;
    rx1-=2f;
    gl.glFlush();

    gl.glFlush();
}
```

EJERCICIO 2



3



4

De igual manera podemos observar que gracias al sombreador “ShadeModel” SMOOTH se puede obtener la interpolación de colores desde cada vértice con los que éstas figuras están creadas.

³ EJERCICIO 2

⁴ EJERCICIO 2

```

public void display(GLAutoDrawable drawable) {
    GL gl = drawable.getGL();

    // Clear the drawing area
    gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    // Reset the current matrix to the "identity"
    gl.glLoadIdentity();

    // Move the "drawing cursor" around
    //gl.glTranslatef(-1.5f, 0.0f, -6.0f);
    gl.glTranslatef(-.7f, -.8f, -5.0f);
    gl.glRotatef(rx,0,1,0);
    gl.glScalef(3,3,3);
    // Drawing Using Triangles

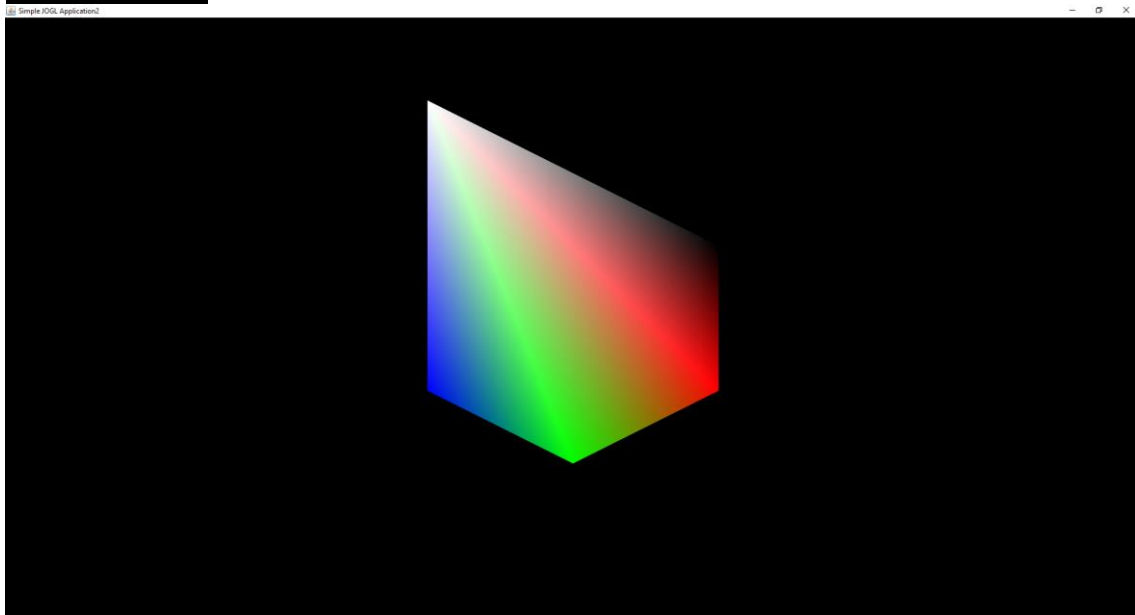
    gl.glBegin (gl.GL_TRIANGLES);
    gl.glColor4f(1,0,0,1F);
    gl.glVertex3f (0, 0, 0);
    gl.glColor4f(0,1,0,1F);
    gl.glVertex3f (.50f, 0, 0);
    gl.glColor4f(0,0,1,1F);
    gl.glVertex3f (.25f, .50f, 0);
    //
    gl.glEnd ();

    gl.glLoadIdentity();
    gl.glTranslatef(-.6f, -.6f, -6.0f);
    gl.glRotatef(rx*.5f,1,0,0);
    gl.glBegin (gl.GL_TRIANGLES);
    gl.glColor4f(0,1,0,1F);
    gl.glVertex3f (0, 0, 0);
    gl.glColor4f(0,0,1,1F);
    gl.glVertex3f (.50f, 0, 0);
    gl.glColor4f(1,0,0,1F);
    gl.glVertex3f (.25f, .50f, 0);
    //
    gl.glEnd ();
    // Flush all drawing operations to the graphics card

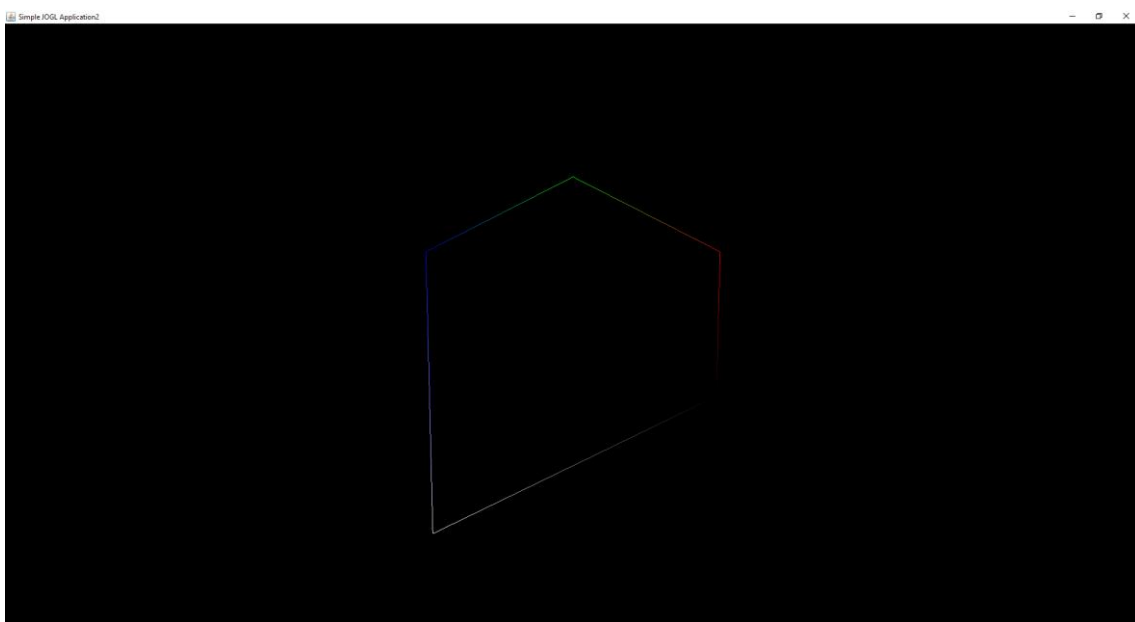
    rx+=1;
    gl.glFlush();
}

```

EJERCICIO 3



5

6

⁵ EJERCICIO 3

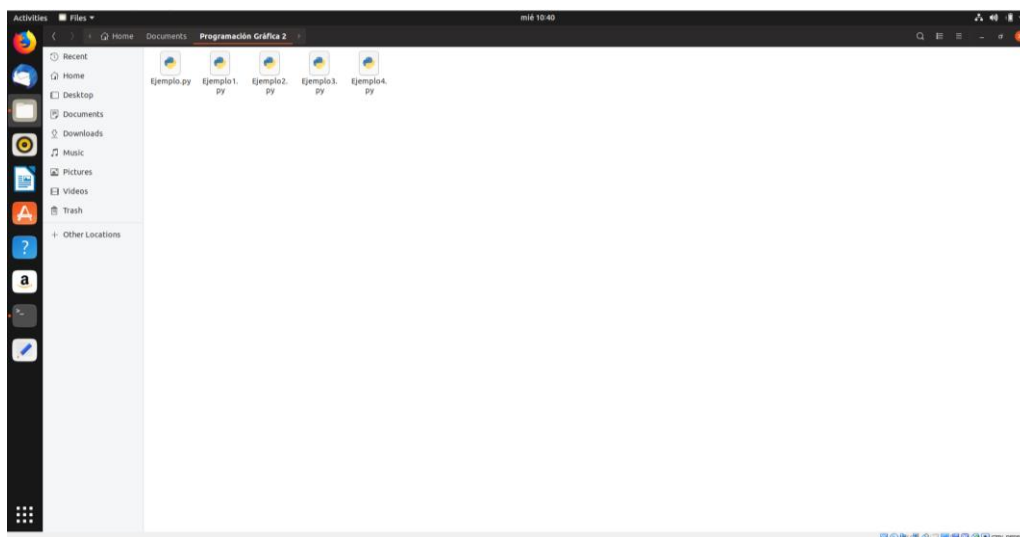
⁶ EJERCICIO 3

En el ejemplo 3 podemos apreciar la interpolación de colores en un polígono construido a partir de 5 vértices, en este ejercicio podemos apreciar la diferencia entre las dos caras del plano, viendo cómo cambia su superficie al rotar, entonces vemos su cara frontal y posterior, cambiando simultáneamente como se indicó antes.

```
public void display(GLAutoDrawable drawable) {
    GL gl = drawable.getGL();
    gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
    gl.glLoadIdentity();
    gl.glTranslatef(0f, 0f, -5.0f);
    gl.glRotatef(rx, 1, 0, 0);
    gl.glScalef(.10f, .10f, .1f);
    gl.glPolygonMode(GL.GL_FRONT, GL.GL_LINE);
    gl.glBegin(GL.GL_POLYGON);
        gl.glColor3f(1.0f, 1.0f, 1.0f);
        gl.glVertex3f(-10, 15, 0);
        gl.glColor3f(0.0f, 0.0f, 0.0f);
        gl.glVertex3f(10, 5, 0);
        gl.glColor3f(1.0f, 0.0f, 0.0f);
        gl.glVertex3f(10, -5, 0);
        gl.glColor3f(0.0f, 1.0f, 0.0f);
        gl.glVertex3f(0, -10, 0);
        gl.glColor3f(0.0f, 0.0f, 1.0f);
        gl.glVertex3f(-10, -5, 0);
    gl.glEnd();
    gl.glFlush();
    gl.glPolygonMode(GL.GL_BACK, GL.GL_FILL);
    gl.glBegin(GL.GL_POLYGON);
        gl.glColor3f(1.0f, 1.0f, 1.0f);
        gl.glVertex3f(-10, 15, 0);
        gl.glColor3f(0.0f, 0.0f, 0.0f);
        gl.glVertex3f(10, 5, 0);
        gl.glColor3f(1.0f, 0.0f, 0.0f);
        gl.glVertex3f(10, -5, 0);
        gl.glColor3f(0.0f, 1.0f, 0.0f);
        gl.glVertex3f(0, -10, 0);
        gl.glColor3f(0.0f, 0.0f, 1.0f);
        gl.glVertex3f(-10, -5, 0);
    gl.glEnd();
    gl.glFlush()
    //rx+=1F;
    // Flush all drawing operations to the graphics card
    gl.glFlush();
}
```

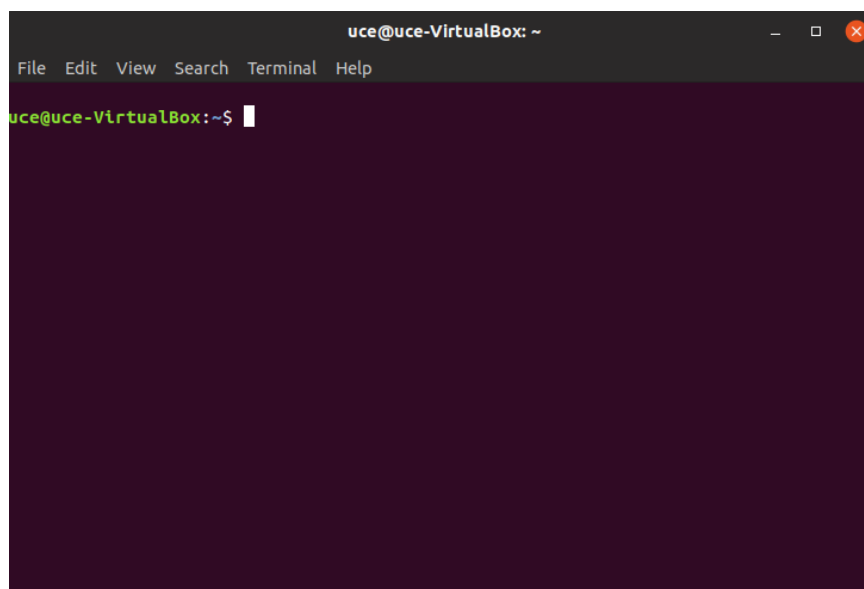
EJERCICIOS PYTHON

Para empezar a realizar ejercicios, es necesario saber que se necesita la librería pygame, py.game y un editor de notas, es necesario tener en cuenta que se necesita la extensión, y al terminar el código indexado se aplica la extensión “.py” y se accede a la línea de comandos, consiguientemente a la carpeta donde se encuentra el programa y ejecutarlo, a continuación se presenta un ejemplo de cómo ejecutar un programa.



7

En la imagen 1 se muestran programas que se encuentran en la carpeta “Programación Gráfica 2”, para acceder a ellos, al abrir una línea de comando se presentará una ventana como la de la imagen 2.



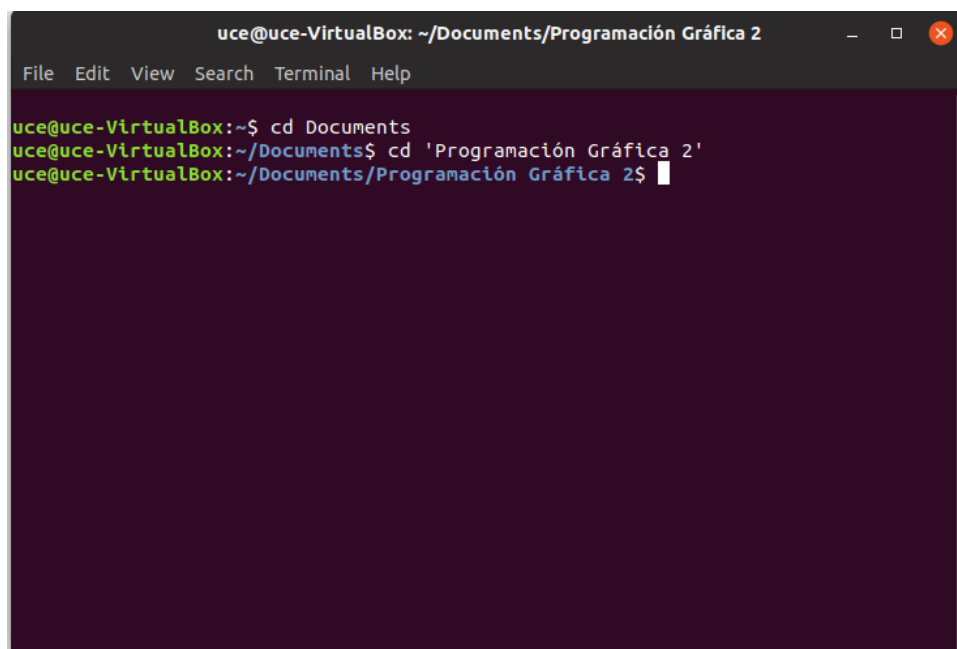
8

A continuación ingresamos a la carpeta, con el comando “*cd Documents*”, después a la carpeta “Programación Gráfica 2” con el comando “*cd 'Programación Gráfica 2'*” y se

⁷ REFERENCIA DE LOCALIZACIÓN DE ARCHIVOS

⁸ LÍNEA DE COMANDOS

accederá a la carpeta que contiene los Ejemplos de programas, como se muestra en la imagen 3.



```

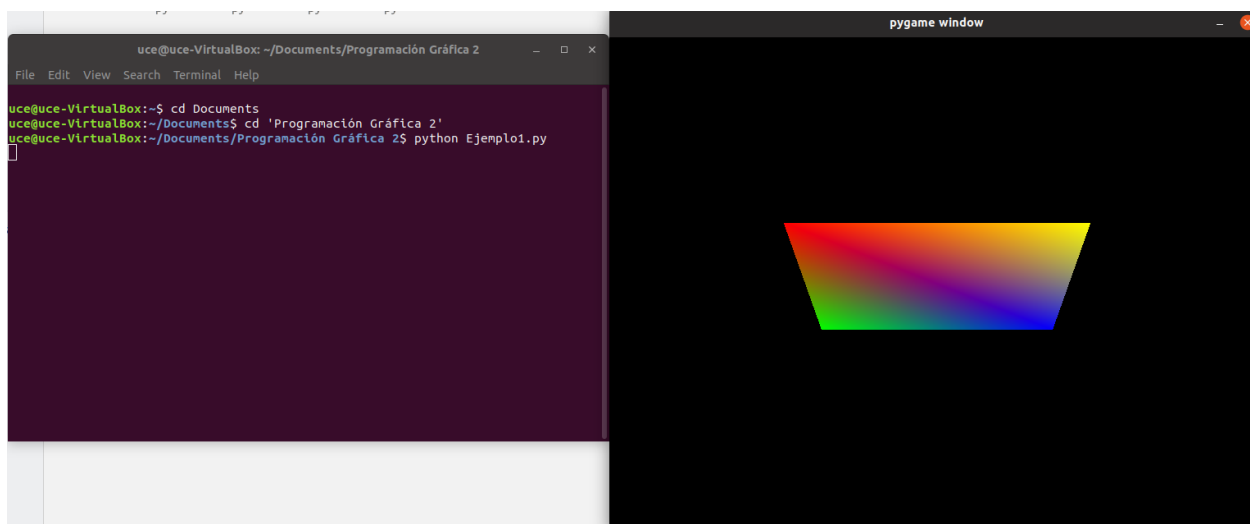
uce@uce-VirtualBox: ~/Documents/Programación Gráfica 2
File Edit View Search Terminal Help

uce@uce-VirtualBox:~$ cd Documents
uce@uce-VirtualBox:~/Documents$ cd 'Programación Gráfica 2'
uce@uce-VirtualBox:~/Documents/Programación Gráfica 2$

```

9

A continuación ingresamos a la carpeta, con el comando “*cd Documents*” después a la carpeta “Programación Gráfica 2” con el comando “*cd 'Programación Gráfica 2'*” y finalizamos con el comando: “*python Ejemplo1.py*”

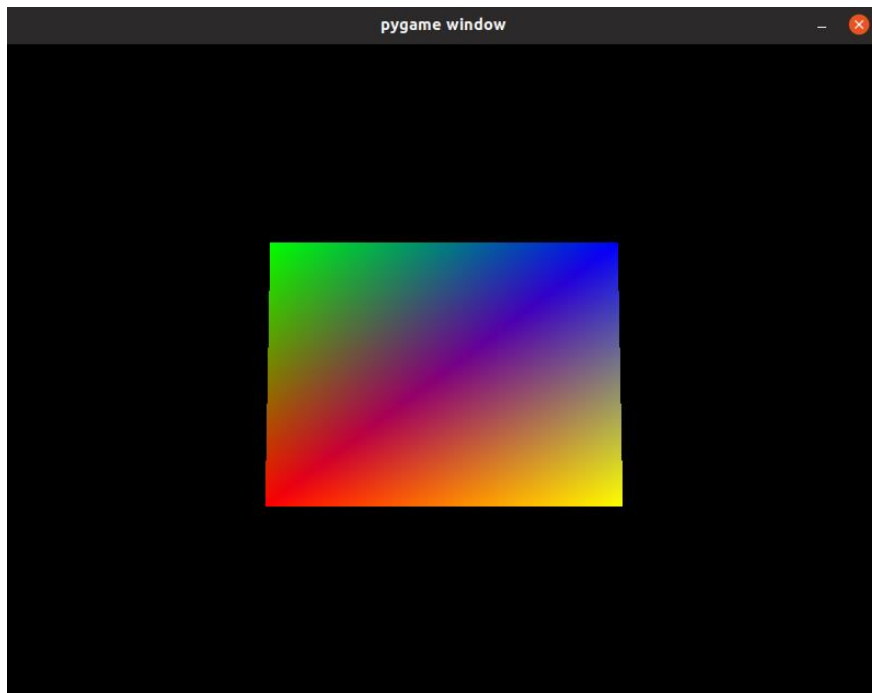


10

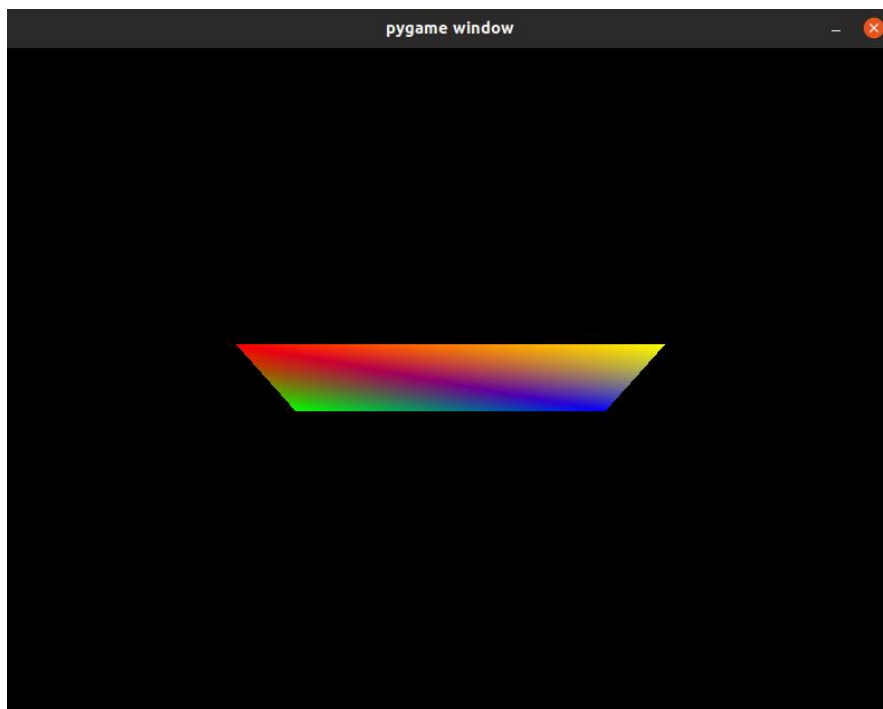
Como podemos observar, se compilará el código del programa al que accedimos, de igual manera se realiza para compilar otro programa. En las siguientes páginas observaremos los ejercicios realizados en Python y su código.

⁹ LÍNEA DE COMANDO PARA INGRESAR A UNA CARPETA ESPECÍFICA.

¹⁰ EJECUCIÓN DE UN PROGRAMA

EJERCICIO 1:

11



12

```
import pygame  
from pygame.locals import *
```

¹¹ EJERCICIO1

¹² EJERCICIO1

```

from OpenGL import *
from OpenGL.GL import *
from OpenGL.GLU import *

def main():
    ry=1;
    pygame.init()
    display=(800,600)
    pygame.display.set_mode(display, DOUBLEBUF|OPENGL)
    gluPerspective(45, (display[0]/display[1]),0.1,150)
    glTranslatef(0,0,-6)
    while True:
        for event in pygame.event.get():
            if event.type==pygame.QUIT:
                pygame.quit()
                quit()

        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)

        glRotatef(1,1,0,0)

        glBegin(GL_POLYGON)

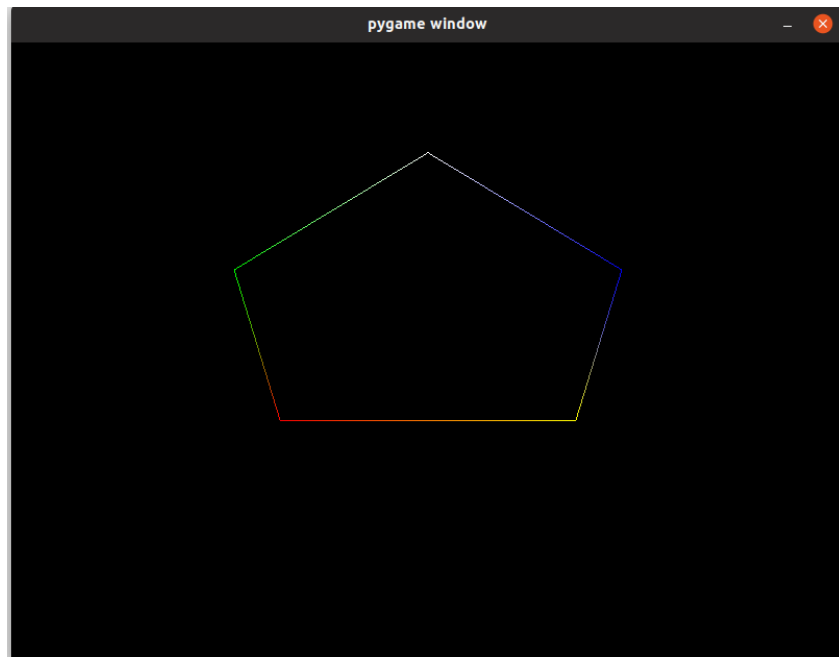
        glColor3f(1,0,0)
        glVertex3f(-1,-1,0)
        glColor3f(0,1,0)
        glVertex3f(-1,1,0)
        glColor3f(0,0,1)
        glVertex3f(1,1,0)
        glColor3f(1,1,0)
        glVertex3f(1,-1,0)

        glEnd()

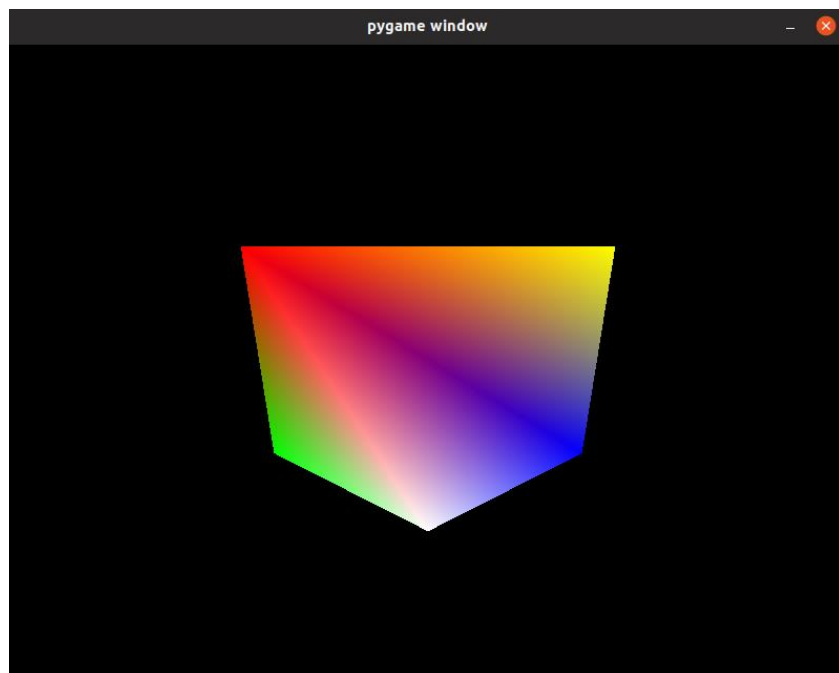
        pygame.display.flip()
        pygame.time.wait(10)

    ry+=1
main()

```

EJERCICIO 2:

13



14

¹³ EJERCICIO 2¹⁴ EJERCICIO 2

```

import pygame
from pygame.locals import *
from OpenGL import *
from OpenGL.GL import *
from OpenGL.GLU import *

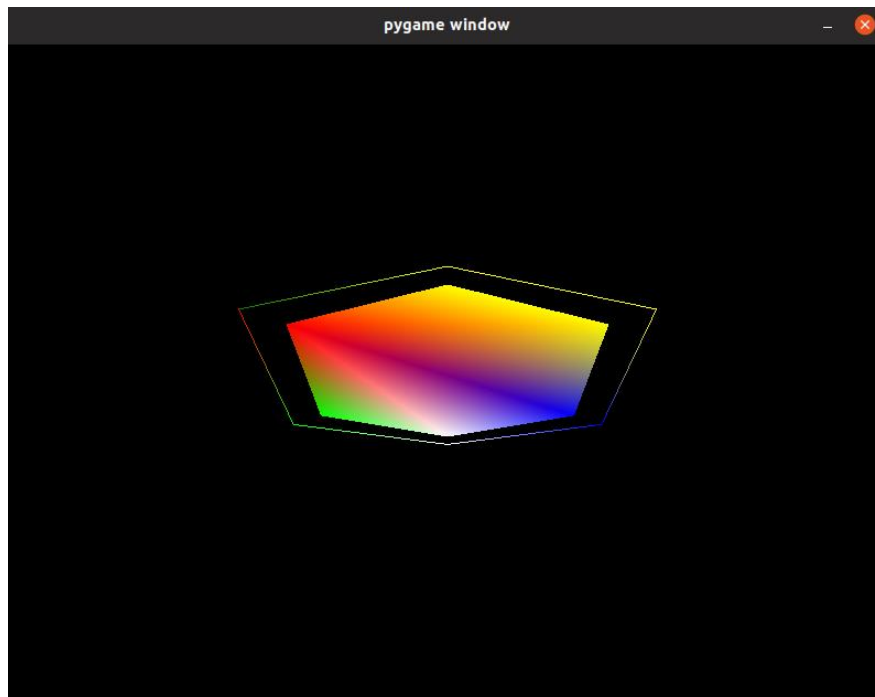
def main():
    pygame.init()
    display=(800,600)
    pygame.display.set_mode(display, DOUBLEBUF|OPENGL)
    gluPerspective(45, (display[0]/display[1]),0.1,150)
    glTranslatef(0,0,-6)
    while True:
        for event in pygame.event.get():
            if event.type==pygame.QUIT:
                pygame.quit()
                quit()

        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
        glRotatef(1,1,0,0)
        glPolygonMode(GL_FRONT, GL_FILL)
        glBegin(GL_POLYGON)
        glColor3f(1,0,0)
        glVertex2f(-1,-1)
        glColor3f(0,1,0)
        glVertex2f(-1,1)
        glColor3f(1,1,1)
        glVertex2f(0,2)
        glColor3f(0,0,1)
        glVertex2f(1,1)
        glColor3f(1,1,0)
        glVertex2f(1,-1)
        glEnd()
        glFlush()

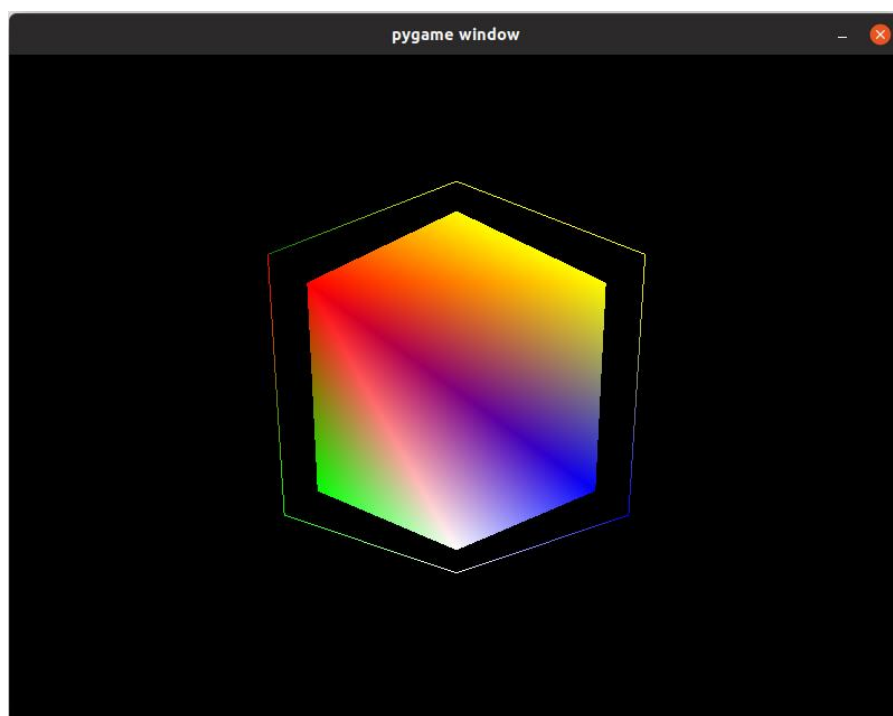
        glPolygonMode(GL_BACK, GL_LINE)
        glBegin(GL_POLYGON)
        glColor3f(1,0,0)
        glVertex2f(-1,-1)
        glColor3f(0,1,0)
        glVertex2f(-1,1)
        glColor3f(1,1,1)
        glVertex2f(0,2)
        glColor3f(0,0,1)
        glVertex2f(1,1)
        glColor3f(1,1,0)
        glVertex2f(1,-1)
        glEnd()
        glFlush()
        pygame.display.flip()
        pygame.time.wait(10)

main()

```

EJERCICIO 3

15



16

¹⁵ EJERCICIO 3

¹⁶ EJERCICIO 3

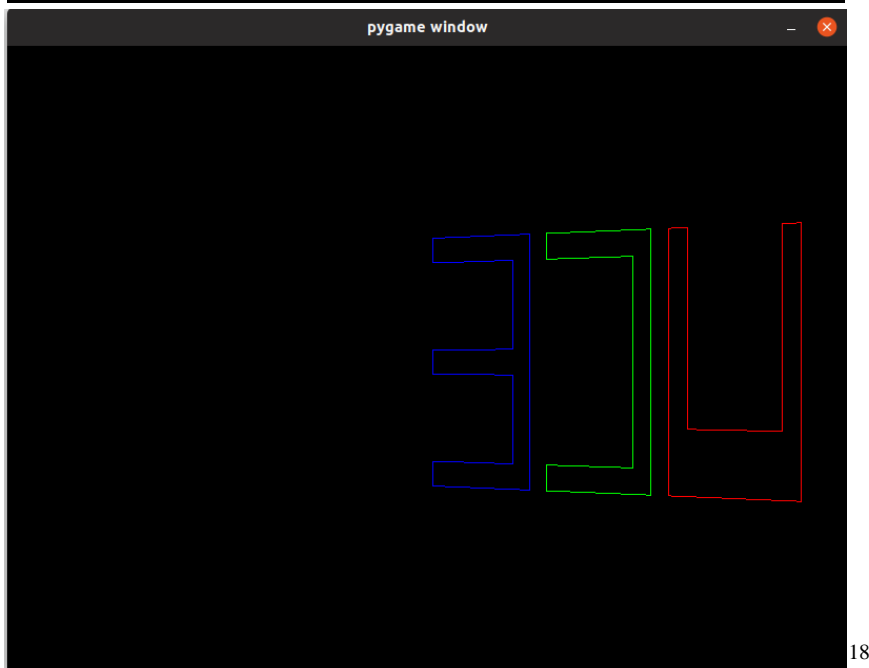
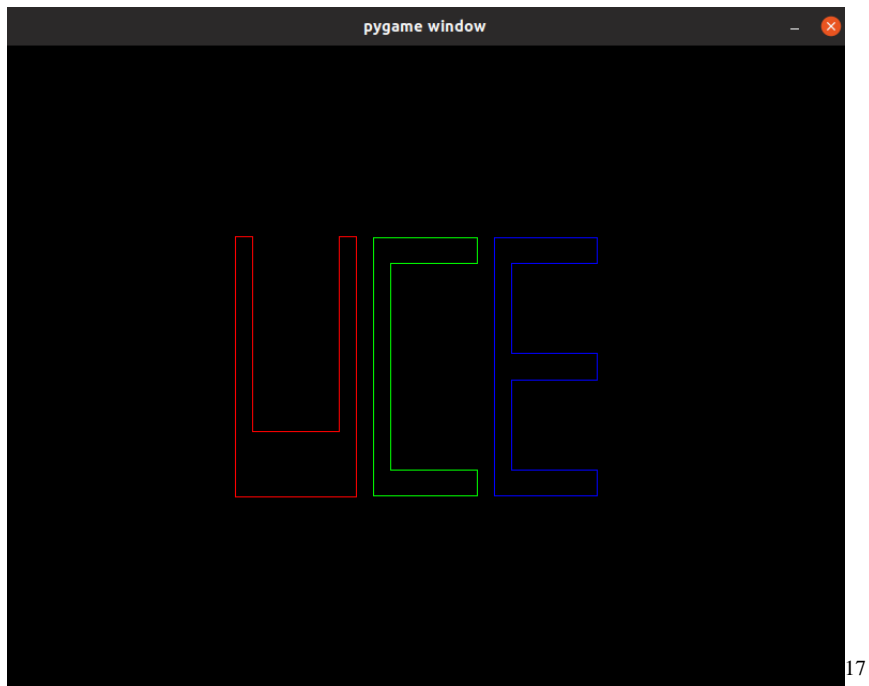
```

import pygame
from pygame.locals import *
from OpenGL import *
from OpenGL.GL import *
from OpenGL.GLU import *

def main():
    pygame.init()
    display=(800,600)
    pygame.display.set_mode(display, DOUBLEBUF|OPENGL)
    #glShadeModel(GL_FLAT)
    glShadeModel(GL_SMOOTH)
    gluPerspective(45, (display[0]/display[1]),0.1,150)
    glTranslatef(0,0,-6)
    while True:
        for event in pygame.event.get():
            if event.type==pygame.QUIT:
                pygame.quit()
                quit()
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
        glRotatef(1,1,0,0)
        glBegin(GL_LINE_STRIP)
        glColor3f(1,0,0)
        glVertex2f(-1,-1)
        glColor3f(0,1,0)
        glVertex2f(-1,1)
        glColor3f(1,1,1)
        glVertex2f(0,1.5)
        glColor3f(0,0,1)
        glVertex2f(1,1)
        glColor3f(1,1,0)
        glVertex2f(1,-1)
        glColor3f(1,1,0)
        glVertex2f(0,-1.5)
        glColor3f(0,.5,0)
        glVertex2f(-1,-1)
        glEnd()
        glBegin(GL_POLYGON)
        glColor3f(1,0,0)
        glVertex2f(-.8,-.8)
        glColor3f(0,1,0)
        glVertex2f(-.8,.8)
        glColor3f(1,1,1)
        glVertex2f(0,1.3)
        glColor3f(0,0,1)
        glVertex2f(.8,.8)
        glColor3f(1,1,0)
        glVertex2f(.8,-.8)
        glColor3f(1,1,0)
        glVertex2f(0,-1.3)
        glColor3f(0,.5,0)
        glVertex2f(-.8,-.8)
        glEnd()
        pygame.display.flip()
        pygame.time.wait(10)

main()

```

EJERCICIO 4

¹⁷ EJERCICIO 4

¹⁸ EJERCICIO 4

```

import pygame
from pygame.locals import *
from OpenGL import *
from OpenGL.GL import *
from OpenGL.GLU import *

def main():
    pygame.init()
    display=(800,600)
    pygame.display.set_mode(display, DOUBLEBUF|OPENGL)
    #glShadeModel(GL_FLAT)
    glShadeModel(GL_SMOOTH)
    gluPerspective(45, (display[0]/display[1]),0.1,150)
    glTranslatef(0,0,-6)
    while True:
        for event in pygame.event.get():
            if event.type==pygame.QUIT:
                pygame.quit()
                quit()
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
        glRotatef(1,1,0,0)
        glBegin(GL_LINE_STRIP)
        glColor3f(1,0,0)
        glVertex2f(-1,-1)
        glColor3f(0,1,0)
        glVertex2f(-1,1)
        glColor3f(1,1,1)
        glVertex2f(0,1.5)
        glColor3f(0,0,1)
        glVertex2f(1,1)
        glColor3f(1,1,0)
        glVertex2f(1,-1)
        glColor3f(1,1,0)
        glVertex2f(0,-1.5)
        glColor3f(0,.5,0)
        glVertex2f(-1,-1)
        glEnd()

        glBegin(GL_POLYGON)
        glColor3f(1,0,0)
        glVertex2f(-.8,-.8)
        glColor3f(0,1,0)
        glVertex2f(-.8,.8)
        glColor3f(1,1,1)
        glVertex2f(0,1.3)
        glColor3f(0,0,1)
        glVertex2f(.8,.8)
        glColor3f(1,1,0)
        glVertex2f(.8,-.8)
        glColor3f(1,1,0)
        glVertex2f(0,-1.3)
        glColor3f(0,.5,0)
        glVertex2f(-.8,-.8)
        glEnd()
        pygame.display.flip()
        pygame.time.wait(10)

main()

```


Lista de referencias

- American Psychological Association. (2010). *Manual de Publicaciones de la American Psychological Association* (6 ed.). (M. G. Frías, Trad.) México, México: El Manual Moderno.
- Atlassian Bitbucket. (2018). *Atlassian Bitbucket*. Obtenido de Atlassian Bitbucket: <https://www.atlassian.com/git/tutorials/undoing-changes/git-rm>
- Aulas / Uruguay Educa. (19 de Abril de 2019). Obtenido de Ciencias Físicas - 1º C.B.: <http://aulas.uruguayeduca.edu.uy/mod/book/view.php?id=23929&chapterid=5872>
- Bender. (2019). *VIX*. Obtenido de <https://www.vix.com/es/btg/curiosidades/2010/12/13/la-teoria-del-color>
- Cloud Computing. (2019). *Cloud Computing*. Obtenido de Cloud Computing: <http://iesgn.github.io/cloud/cursos/u2/git>
- Desarrollo Web. (16 de Noviembre de 2016). *Desarrollo Web*. Obtenido de Desarrollo Web: <https://desarrolloweb.com/articulos/archivo-gitignore.html>
- Estrada, E. (2018). *FotoNostra*. Obtenido de <https://www.fotonostra.com/grafico/teoriacolor.htm>
- Git. (2019). *Git*. Obtenido de Git: <https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones>
- GitLab Docs. (2019). *GitLab Docs*. Obtenido de GitLab Docs: <https://docs.gitlab.com/ee/ssh/>
- Molina Carmona, R., & Puchol García, J. A. (1999). *Departamento de Informática*. Obtenido de Universidad Nacional de San Luis: http://www.dirinfo.unsl.edu.ar/servicios/abm/assets/uploads/materiales/1080c-05_apuntes_opengl.pdf
- Shinners, P. (2000). *pygame*. Obtenido de <https://www.pygame.org/docs/tut/PygameIntro.html>
- Shinners, P. (2000). *pygame*. Obtenido de https://www.pygame.org/docs/ref/color.html#comment_pygame_Color
- Sociedad de Prevención de FREMAP. (2013). Obtenido de <http://www.iqm.csic.es/wp-content/uploads/2013/prevencion/recomendaciones%20especificas/14.ILUMINACION.pdf>

