Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Ingeniería en Computación

Reporte Final del Proyecto Terminal

“Modificación del algoritmo de trazado de rayos para simular deformaciones 3D y generar ambientes surreales”

Trimestre 10O

Alumno:

Vázquez Leal Edgar Ismael

206200888

Asesor:

Risto Rangel Kuoppa

Tabla de contenido

Introducción

El presente documento refleja el desarrollo del PT de la propuesta exponiendo el trabajo realizado durante cada etapa de la misma así como los resultados en términos de una encuesta realizada a (algunas) personas. Las encuestas así como la interpretación de los resultados se anexan al final del documento.

La primera parte del proyecto implica crear una aplicación que sea capaz de implementar el algoritmo de trazado de rayos, es decir, que permita crear ambientes a base de primitivas, las cuales deberán ser implementadas a través de algoritmos de intersección.

Cuando se tenga implementado el algoritmo de trazado de rayos lo que sigue es diseñar las modificaciones para que los rayos del algoritmo anterior no sigan una trayectoria recta, sino curva. Este punto se trabajó utilizando curvas paramétricas y un método de interpolación.

En la parte final se agrega, luminosidad y textura a los ambientes geneerados. Para comprobar si el objetivo se ha cumplido se debe aplicar una encuesta de percepción, es decir, anónima y dónde simplemente se pregunta de acuerdo a cada encuestado su opinión acerca de las escenas generadas.

Justificación

Actualmente el campo de las gráficas por computadora, ha tomado un gran interés desde hace ya varios años lo que ha desencadenado en una contínua e imparable evolución del área con el mejoramiento en materia de hardware y en las investigaciones sobre técnicas de representaciones gráficas.**[1]**

Ilustración 2 "Sleep" por Salvador Dali

Ilustración 1"La firma en blanco" por Rene Magritte

Aunque la tendencia de las tecnologías tanto de hardware como de software para el procesamiento de imágenes por computadora es lograr imágenes más realistas o hasta mejorarlas, siempre está presente el gusto por lo surreal[1]. Ejemplos de este género los tenemos en pinturas como las de Salvador Dali o Rene Magritte, ejemplos de sus obras se muestran en las Figuras 1 y 2.

El surrealismo es una expresión artística utilizada en literatura, artes plásticas y más recientemente en los medios audiovisuales tales como la industria cinematográfica, en la que destacan directores como David Lynch y Jean-Pierre Jaunet, entre otros.**[2]**

La técnica de trazado de rayos es un método de *renderizado* [2]directo que hace uso de un modelo de iluminación semiglobal, pues además de considerar la contribución de las fuentes de luz, también tiene en cuenta la luz reflejada y emitida por los objetos circundantes**[4]**.

Para generar las deformaciones en los objetos nos enfocaremos en modificar la forma en que el algoritmo genera las imágenes al modificar las trayectorias de los rayos. Modificando la estructura de los rayos cambiará la proyección del objeto en la imagen final, es decir, se obtendrán efectos de distorsión de los objetos (como en el arte surrealista) en la escena pero sin modificar la geometría que define a la superficie de los objetos. Estas modificaciones se busca que generen efectos de distorsiones en la superficie como las que se muestra en algunas de las pinturas de los artistas surrealistas. Figura 3.

Ilustración 3"Galatea Of The Spheres" de Salvador Dali

[1] Visión bizarra o de ensueño de las cosas
[2] Creación de una imagen por computadora

Objetivo General

Modificar el algoritmo de trazado de rayos e implementarlo en un programa que genere imágenes de escenas 3D para simular deformaciones de los objetos en las escenas.

Objetivos Particulares

- Sintetizar el algoritmo de trazado de rayos
- Implementar el algoritmo de trazado de rayos
- Diseñar modificaciones al algoritmo de trazado de rayos para que éstos sigan trayectorias distintas a líneas rectas
- Implementar el algoritmo modificado usando curvas paramétricas de primer, segundo, y tercer grado para las trayectorias de los rayos
- Realizar una encuesta a un grupo de voluntarios para validar las imágenes obtenidas con el programa como surreales y reportar los resultados

Desarrollo

Existen dos tipos de trazados de rayos: trazado de rayos hacia adelante y trazado de rayos hacia atrás. La diferencia entre ellos está en la facilidad de implementación, es por esta razón que se había elegido inicialmente el trazado de rayos hacia adelante para este proyecto.

Inmediatamente al implementar el trazador de rayos hacia adelante tuvimos algunos problemas en cuanto a rendimiento, es decir, tardaba mucho en renderizar una imagen de manera normal y esto no era factible para crear una aplicación interactiva ya que la cantidad de procesamiento iba a aumentar al agregar las curvas paramétricas. Debido a lo anterior se decidió cambiar al algoritmo de trazador de rayos hacia atrás.

El trazado de rayos hacia atrás funciona de la misma manera en la que el Sol ilumina los objetos sobre la superficie de la Tierra. Se tiene una fuente de luz que lanza rayos hacia la superficie, cuando estos rayos golpean algún punto en un objeto sobre la superficie se obtiene la información de color del punto en cuestión y se renderiza si es el más cercano hacia el observador. La figura 4 ilustra el algoritmo de trazado de rayos hacia atrás.

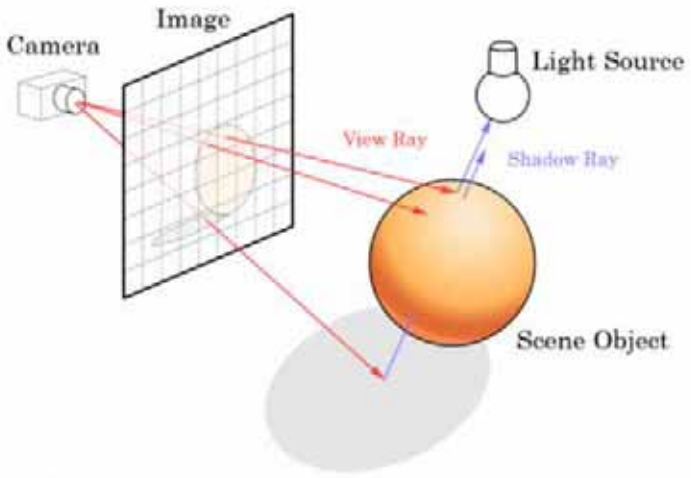

Ilustración 4. Trazado de rayos hacia atrás (backward ray tracing)

Con el algoritmo tuvimos una mejor eficiencia al renderizar imágenes de manera normal. Y al implementar las curvas paramétricas no tuvimos problemas de rendimiento.

Se crearon decenas de imágenes con distintos puntos de control y distintas geometrías. Al final se decidió recrear un efecto de cualquier pintura surreal. Lo cual se implementó con una curva paramétrica de tercer grado con todos sus puntos de control diferentes. Sumado a esto se implementó una división en la pantalla lo que provocó que al hacer la interpolación entre los rayos el cambio de un punto a otro no fuera inmediato.

Al final creamos una imagen "normal" y después con la misma geometría de esa imagen aplicamos el trazador con curvas paramétricas. Como vemos en este primer intento dónde la figura 5 representa la imagen obtenida con el trazador de rayos sin modificación y la siguiente con el trazador de rayos modificado.

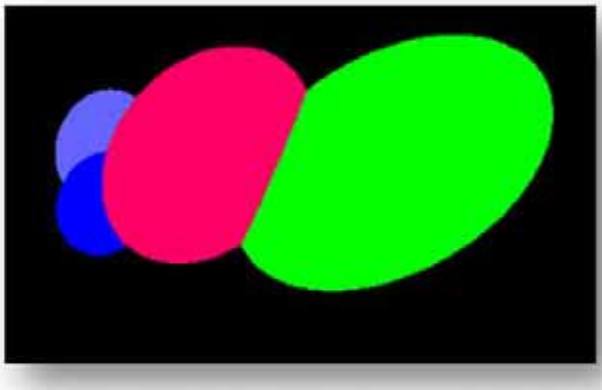

Ilustración 5. Imagen renderizada con el trazador de rayos sin modificaciones

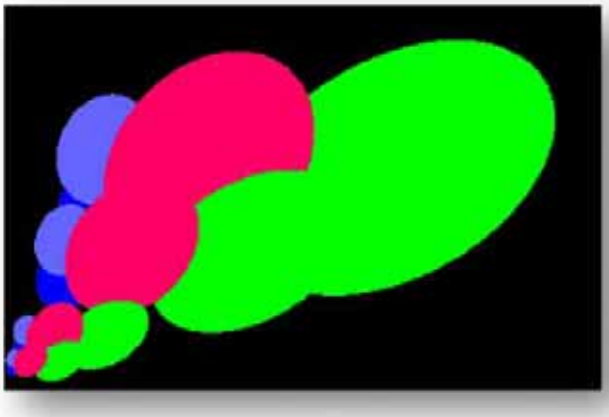

Ilustración 6. Imagen renderizada con el trazador de rayos modificado usando una curva paramétrica de tercer grado

Después de lograr la deformación en las imágenes se buscó una pintura de Salvador Dali de la que pudiéramos extraer algún efecto. La Ilustración 7 muestra la pintura elegida.

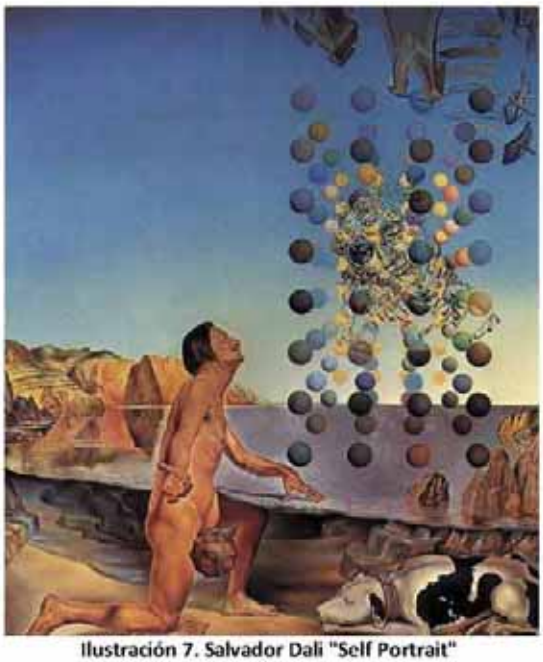

Ilustración 7. Salvador Dali "Self Portrait"

La elección de la pintura se debió principalmente al efecto de las esferas flotantes, ya que se cuentan con las primitivas para trazar una imagen similar con la aplicación. A partir de esta decisión comenzamos a hacer pruebas para intentar recrear el efecto de las esferas. La ilustración 8 muestra la versión del trazador de rayos hacia atrás sin modificaciones. Y la 9 la versión creada nuestro trazador modificado.

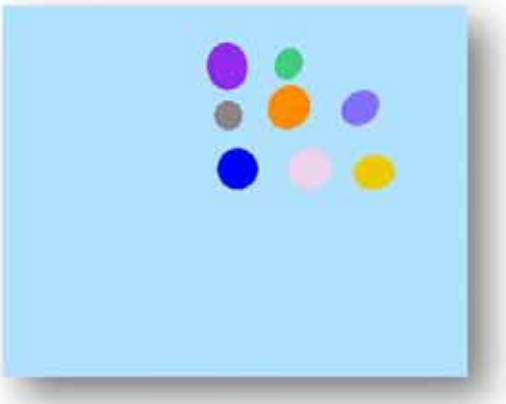

Ilustración 8. Escena fuente con la que se intenta crear el efecto de las esferas.

Ilustración 9. Creada con las mismas primitivas que la ilustración 8 pero con el trazador de rayos modificado

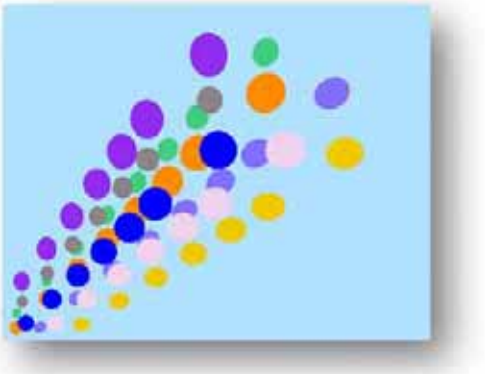

Ilustración 10. Creada con distintos puntos de control

A nuestro criterio obtuvimos un resultado aceptable, por lo que continuamos probando con otras geometrías y modificando los puntos de control de la curva paramétrica. Partimos de un triángulo en el espacio como se muestra en la ilustración 11.

Ilustración 11. Triángulo creado con el trazador de rayos sin modificación

Además de implementar cambios en los puntos de control se creó una función dentro de la aplicación que va cambiando el punto de control no por las intersecciones sino por el número de aumentos en el eje Y. Es decir, se renderizan cierto número de "renglones" de pixeles por así decirlo con un punto de control y así sucesivamente. Dependiendo de la distancia entre cada punto los efectos que se pueden crear. La ilustración 12 se creó cambiando de punto de control cada renglón, es por eso que parece que algunos son del color del fondo pues el punto de control asignado para ese renglón en particular no encontraba intersecciones con la figura.

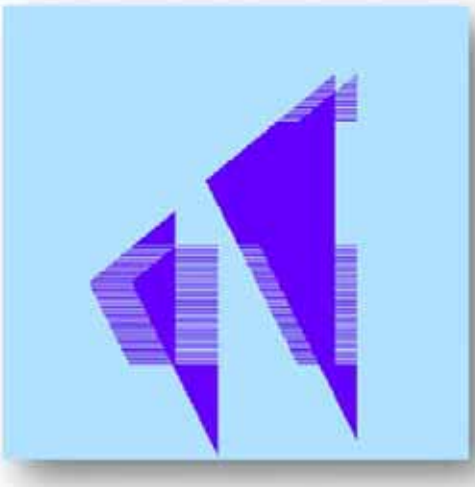

Ilustración 12. Figura creada al hacer cambio en el punto de control en cada aumento en el eje Y

El siguiente fragmento muestra el efecto que se consigue al aumentar el número de renglones por punto de control. En este caso se divide la pantalla en 6 partes.

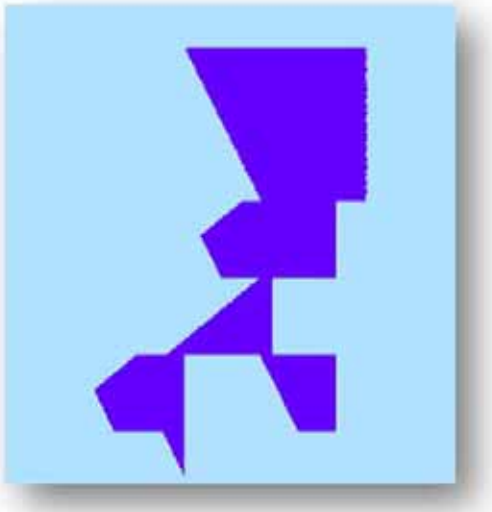

Ilustración 13. Se hace cambio de punto de control 6 veces (cada 150 niveles de eje Y).

Al observar que el efecto creado por esta función era bastante perceptible, se continúo haciendo pruebas con el mismo triángulo pero ahora variando los puntos de control.

Ilustración 14. Creada a partir de variar los puntos de control de la curva paramétrica de la ilustración anterior.

En la imagen siguiente se muestra el triángulo al cambiar el punto de control cada 60 niveles del eje Y. Así mismo se modificaron los puntos de control para intentar dejar claro que se trata del mismo triángulo pero visto a distintas profundidades.

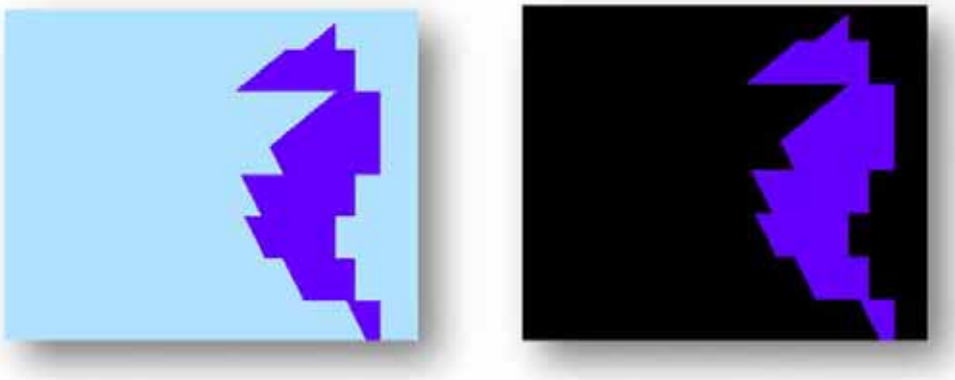

Ilustración 15. Versión final de la modificación de la ilustración 11 y su versión en fondo negro.

A partir de la versión en fondo negro modificamos la función del cambio de punto de control para que se redujera el número de niveles que traza cada punto de control. Los resultados fueron los siguientes.

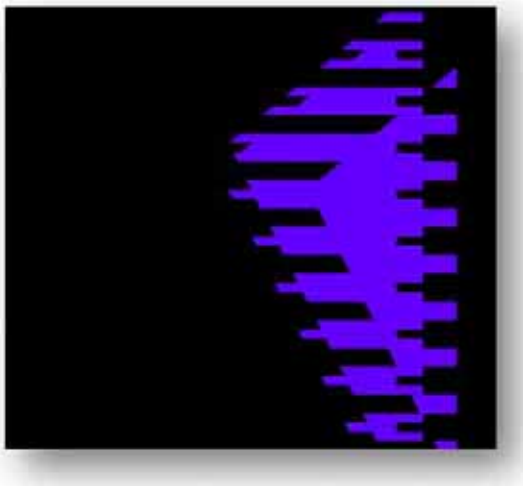

Ilustración 16. Cambio de pundo de control cada 30 niveles de eje Y

En la imagen anterior se duplicó el número de divisiones que se crean en pantalla. En la siguiente el número de divisiones aumentó aún más.

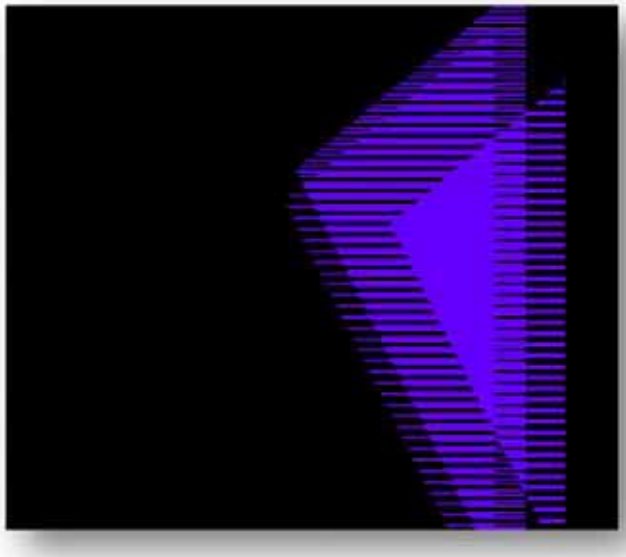

Ilustración 17. Cambio de punto de control cada 10 niveles de eje Y

Esta última imagen fue creada al hacer un cambio de punto de control por cada índice del eje Y, es decir, se tienen 480 cambios. Y se obtiene una imagen como esta:

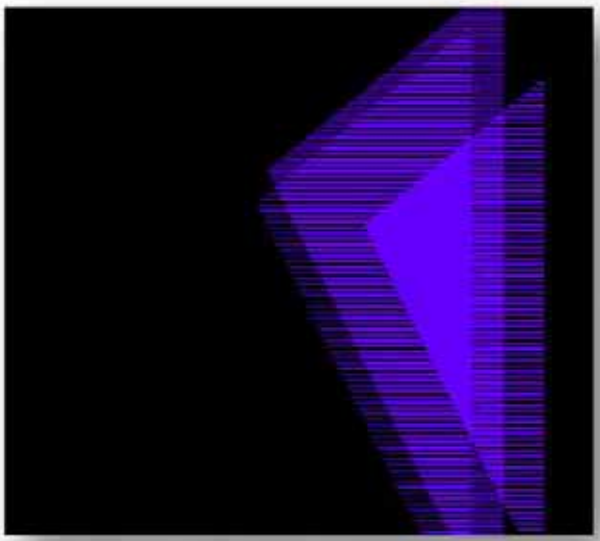

Ilustración 18. 480 cambios de punto de control

Los efectos obtenidos son bastante agradables así que decidimos seguir con el ejercicio creativo y cambiar las primitivas que se trazaban. Utilizamos esferas de distintos tamaños y profundidades. También se modificaron los puntos de control para que hubiera mayor distancia entre ellos, lo que provoca que las formas adquieran un efecto de alargamiento.

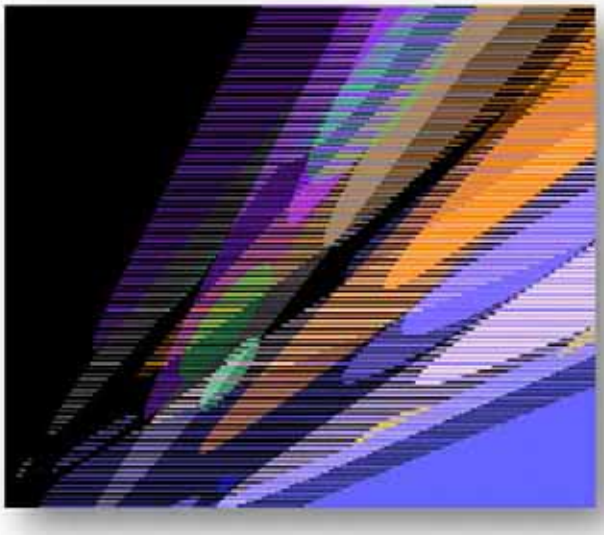

Ilustración 19. 480 cambios de punto de control con esferas

Ya pensando en la imagen que se utilizaría para la encuesta de valoración, se agregaron más primitivas a la escena para aumentar la complejidad de esta y de nuevo se modificaron los puntos de control, esto último como parte del ejercicio creativo de probar efectos.

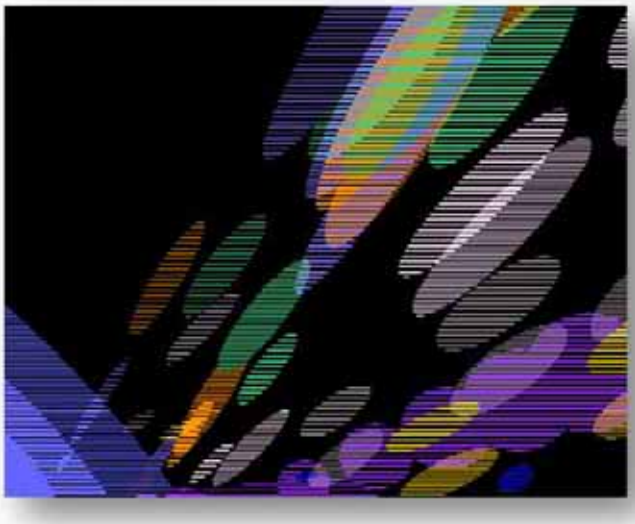

Ilustración 20. 480 cambios de punto de control con más esferas

Al aplicar las funciones de intersección de las primitivas con los rayos, fue más complicado de lo que teníamos contemplado, por lo que se ocupó tiempo extra del que se tenía planeado y no se implementó la iluminación y las texturas en dichas geometrías.

Trabajando con la complejidad necesaria para aplicar la encuesta generamos la siguiente imagen para valorarla mediante la encuesta. Después se eligieron dos pinturas de Salvador Dalí para formular de manera adecuada las preguntas de la encuesta.

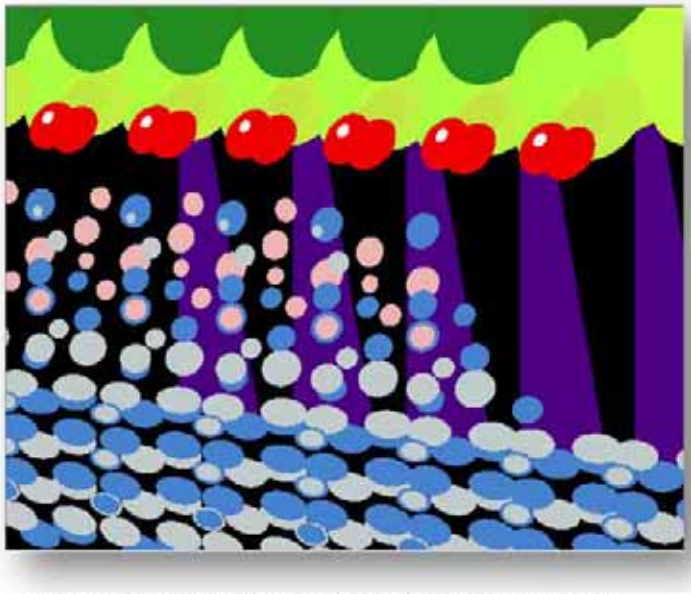

Ilustración 21. Imagen final generada por el trazador modificado

Bibliografía

[1]. ABI-CHAHLA, Feby. *OpenGL 3 & DirectX 11: The war is ove* [en línea]. Septiembre 16 de 2008 [ref. de Febrero 4 de 2010]. Disponible en: <http://www.tomshardware.com/reviews/opengl-directx,2019.html>

[2]. WIKIPEDIA.ORG. *Surrealismo* [en línea]. Noviembre 2009 [ref. de Febrero 4 de 2010]. Disponible en: < http://es.wikipedia.org/wiki/Surrealismo#La_pintura_surrealista>

[3]. ALLEN, Sherrod. *Game Graphics Programming*. Estados Unidos de América: Course of Technology CENGAGE Learning, 2008. ISBN.

[4]. RAMOS, Ricardo. *Tema V: Trazado de Rayos (ray tracing)*. [ref. de Febrero 4 de 2010]. Disponible en: < http://www.di.uniovi.es/~rr/Tema5.pdf>

[5]. WIKIPEDIA.ORG. *Trae Vision TGA.* [en línea]. Noviembre 2009 [ref. de Febrero 11 de 2010]. Disponible en: <http://es.wikipedia.org/wiki/Truevision_TGA>

[6]. WIKIPEDIA.ORG. *DirectX* [en línea]. Febrero 6 de 2010. Disponible en: <http://en.wikipedia.org/wiki/DirectX>

A. Valoración de las imágenes surreales

La encuesta fue realizada a 40 personas, 38 de ellas alumnos de la UAM Azcapotzalco el día 3 de diciembre de 2010. La encuesta contiene una pequeña descripción de lo que se trata el proyecto y acerca de lo que se cuestiona en las preguntas. Tiene un total de 4 preguntas y la quinta es una escala para medir el nivel surreal percibido por el encuestado.

A continuación se presenta una imagen de la encuesta aplicada.

Encuesta

Las imágenes mostradas son parte de un proyecto que busca generar imágenes surreales. Se utiliza el algoritmo de trazado de rayos (hacia atrás) para renderizar las imágenes. Una de ellas se trazó de manera normal y otra modificando el algoritmo para crear un aspecto surreal partiendo de la primera. Cómo parámetro de surrealismo se toman en cuenta pinturas célebres como las del pintor Salvador Dali.

Encargado del proyecto: Edgar Ismael Vázquez Leal email: edgar.vazquez.leal@gmail.com

Nota de privacidad: La información aquí proporcionada se utilizará con fines estadísticos por lo que será totalmente anónima y confidencial.

Después de observar las imágenes circule la opción según su percepción:

1. Tomando en cuenta la imagen 1 como un parámetro de realidad. Considera usted que la imagen 2 es surreal. Sí No
2. Considera que la imagen 2 logra representar algún fragmento de la imagen 3 o 4.
   Sí No
3. A su criterio, considera que la iluminación es la adecuada en las imágenes 1 y 2.
   Sí No
4. Considera que las texturas de las superficies de las imágenes 1 y 2 son buenas, es decir, hay transparencias en los objetos, rugosidad, reflejos, etc.
   Sí No
5. Del 1 al 10, qué grado de surrealismo tiene la imagen 2. Siendo 10 el grado más alto.
   1 2 3 4 5 6 7 8 9 10

Ilustración 22. Encuesta

La siguiente es una captura de la diapositiva que se utilizó para realizar la encuesta. Al principio se pensaba realizar una aplicación para aplicar la encuesta, pero por cuestiones de tiempo se optó por generar la encuesta a manera de presentación para que todos pudieran responder al mismo tiempo y la realización de la misma no tomará más de 10 minutos sin tener que realizar ninguna instalación en los equipos de los encuestados.

Las imágenes se encuentran ordenadas de la siguiente manera:

1. Imagen generada con el trazador de rayos normal.
2. Imagen creada a través del trazador de rayos modificado y cabe mencionar que la geometría en esta imagen es la misma que la utilizada para generar la imagen anterior.
3. Pintura "*Self Portrait 1954*" por Salvador Dalí
4. Pintura "*Butterfly Landscape*" por Salvador Dalí

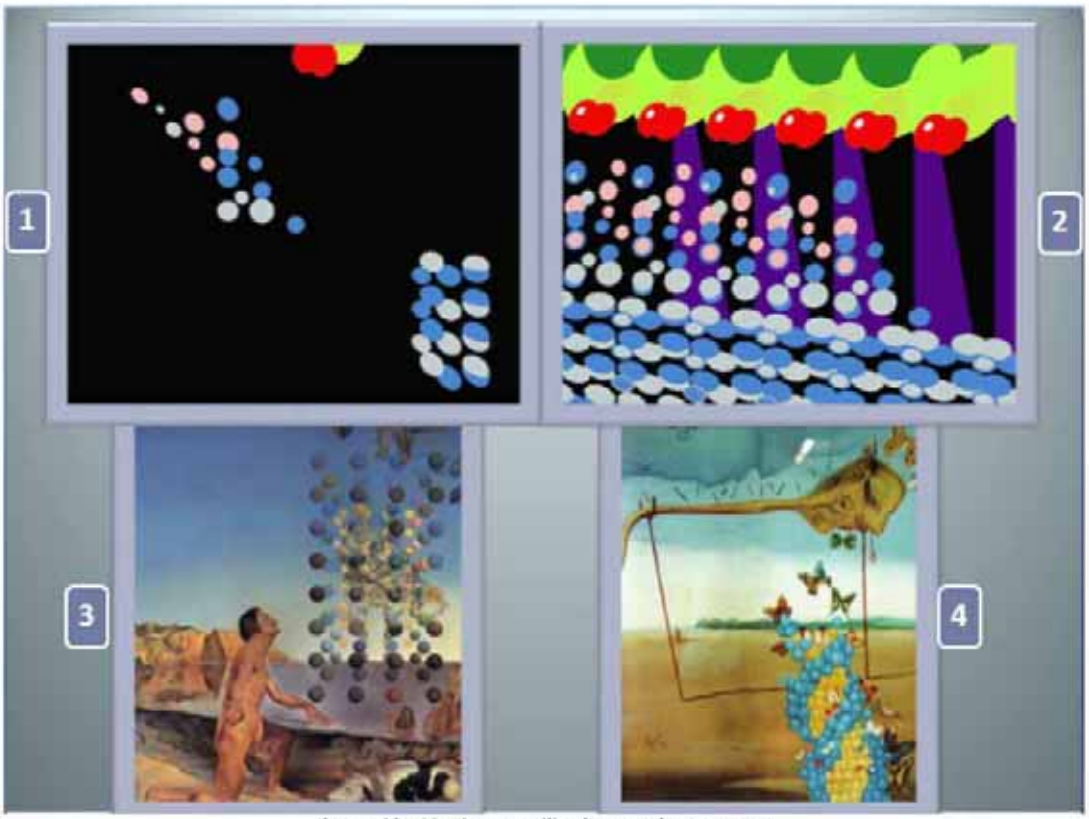

Ilustración 23. Figuras utilizadas para la encuesta

Resultados de la Encuesta

Para un análisis más sencillo y confiable digitalizamos los resultados de cada una de ellas y se muestran en la siguiente tabla. Para un análisis posterior se agruparon las respuestas por número de coincidencias y a partir de allí con los porcentajes obtenidos en cada una de ellas pudimos evaluar las tendencias de los encuestados.

# Encuesta	Pregunta 1.	Pregunta 2.	Pregunta 3.	Pregunta 4.	Pregunta 5.
1	Si	Si	Si	Si	8
2	Si	No	Si	Si	8
3	No	Si	Si	Si	8
4	No	Si	Si	Si	6
5	No	Si	Si	No	6
6	No	Si	Si	Si	4
7	Si	Si	No	No	4
8	No	Si	Si	Si	5
9	Si	No	Si	Si	7
10	No	No	Si	Si	8
11	Si	Si	Si	Si	8
12	Si	Si	Si	Si	8
13	Si	Si	Si	Si	7
14	Si	Si	Si	Si	7
15	No	Si	Si	No	9
16	Si	No	Si	Si	8
17	Si	Si	No	No	7
18	Si	Si	Si	Si	6
19	Si	No	Si	No	8
20	No	No	Si	No	10
21	Si	Si	Si	No	7
22	No	Si	Si	No	2
23	Si	Si	Si	No	8
24	Si	Si	Si	Si	8
25	No	Si	Si	No	6
26	No	Si	Si	No	5
27	No	Si	Si	Si	5
28	Si	Si	Si	No	8
29	Si	Si	Si	Si	8
30	Si	Si	No	No	4
31	Si	Si	No	No	7
32	Si	No	Si	Si	7
33	Si	Si	Si	No	6
34	No	Si	No	Si	6
35	Si	Si	No	Si	4
36	No	Si	Si	Si	3
37	Si	Si	Si	No	8
38	No	Si	No	No	4
39	Si	No	Si	Si	7
40	No	Si	No	No	7

Ilustración 24. Cómputo de los resultados de la encuesta

A partir de la tabla anterior generamos una tabla con los totales que eran necesarios para el análisis de los mismos.

Resultados por pregunta					
	Si	*No*		Si %	No %
Pregunta 1.	24	16		60%	40%
Pregunta 2.	32	8		80%	20%
Pregunta 3.	32	8		80%	20%
Pregunta 4.	22	18		55%	45%

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	**Promedio**
Pregunta 5.	0	1	1	5	3	6	9	13	1	1	**6.55**

Ilustración 25. Resultados agrupados por pregunta

Resultados para el análisis	#	%
Pregunta 1 y 2 satisfactorias.	18	45%
Pregunta 1 y 2 insatisfactorias.	2	5%
Pregunta 3 y 4 satisfactorias.	20	50%
Pregunta 3 y 4 insatisfactorias.	6	15%
Pregunta 5 aprobatoria.	30	75%
Pregunta 5 reprobatoria.	10	25%
Total de Encuestas.	40	100%

Ilustración 26. Resultados a analizar.

Análisis de Resultados

En cuanto a la pregunta 1 en los totales podemos observar que no hay una clara orientación hacia una u otra respuesta. Por lo que podemos concluir que no se logra el objetivo de a partir de una imagen trazada normalmente generar una surreal. Esto puede ser por lo difícil que resulta poner la imagen uno como un parámetro de la realidad y más cuando esta está conformada por solo algunas figuras geométricas.

Por otro lado podemos observar que a los encuestados les fue sencillo identificar algún fragmento de las pinturas en la imagen creada con el trazado de rayos modificado. Esto lo podemos considerar bueno por el trabajo dedicado a generar una imagen en la que fácilmente se pudiera aislar algún efecto y por lo tanto poderla relacionar con una pintura.

En la pregunta 3 también se observa una clara tendencia hacia la respuesta positiva. Las pinturas elegidas, como la mayoría de las generadas en esa época, no tienen grandes efectos luminosos, su grado de dificultad se basaba mayormente en las formas. Ese motivo puede haber impactado en la percepción de los encuestados, por lo que no lograron identificar un parámetro de luminosidad demasiado complejo y por lo tanto consideran que las pinturas generadas tienen luminosidad adecuada.

En la pregunta 4 vemos que provocó indecisión en los encuestados pues casi la mitad estaban en desacuerdo. En mi consideración si falta texturizar las formas en nuestras imágenes generadas, lo que provocaría un efecto más identificable. Los que pudieron estar de acuerdo con que las texturas son suficientes en nuestras imágenes es mayormente porque no hay parámetro de texturas más complejas.

Ahora hablando de la calificación asignada por los encuestados hacía la imagen creada como surreal la tendencia fue positiva, tres cuartas partes de los encuestados valoraron la imagen de manera positiva. Por lo que podemos concluir que a pesar de lo limitado de los recursos que se tenían para generar la imagen, se logró el objetivo ya que los encuestados notaron que era surreal.

Por lo anterior concluimos que el resultado obtenido fue positivo, ya que la mayoría de los encuestados pudo percibir que se reprodujo algún fragmento de las pinturas. El proyecto todavía tiene mucho trabajo pero se consiguió un buen avance. Tal vez, en su continuación sea posible generar imágenes más fiables a las pinturas antes mencionadas y más a futuro se podría crear una aplicación más robusta. Un posible objetivo a futuro sería ese el permitir al usuario manipular algunos parámetros desde una interfaz gráfica para así crear sus propias obras con sus efectos personalizados.

B. Diagrama de Funcionamiento

A continuación se presentan dos diagramas que ilustran la manera en que funciona el sistema de manera interna. Cada uno de los diagramas corresponde a la manera de trazar una imagen ya sea de manera tradicional o con el algoritmo de trazado de rayos modificado.

Para trazar una imagen de manera normal contamos con un buffer de objetos, por objetos entiéndase primitivas de trazado, hasta el momento se tienen dentro de la aplicación esferas, triángulos, planos y cilindros. Se agregan dichos objetos en el buffer, posteriormente se prueba la intersección de los objetos como se describió anteriormente, cabe mencionar que cada objeto tiene una función para encontrar las posibles intersecciones. Los resultados de dichas pruebas se almacenan dentro de un buffer (o arreglo) de colores, en el que si se encuentra intersección se almacena el color del objeto en cuestión, en caso contrario se almacena el color de fondo. Por último el arreglo de colores se almacena como formato de imagen PNG. A continuación el diagrama en la ilustración 27.

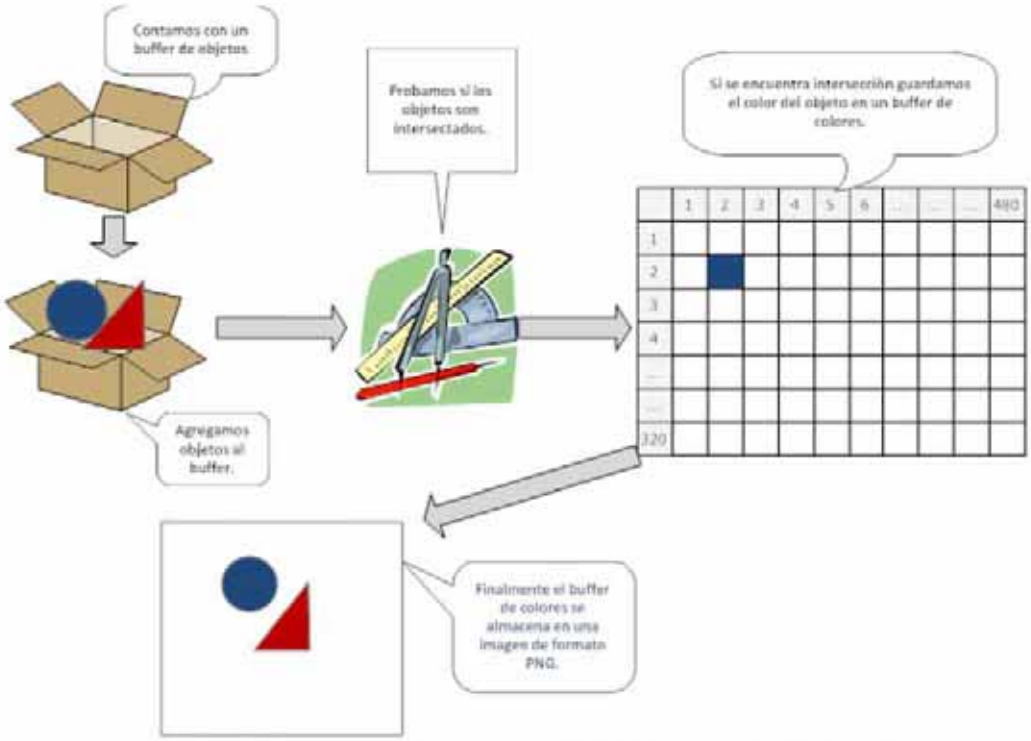

Ilustración 27. Diagrama de funcionamiento del trazador de rayos tradicional.

Para el caso de nuestro algoritmo modificado se agregan algunos pasos que son la creación de una curva paramétrica de tercer grado y la interpolación que se realiza entre cada uno de los puntos de control de mencionada curva. La interpolación nos arroja pequeños segmentos de línea que serán utilizados como rayos para probar las intersecciones con los objetos en nuestro buffer.

Cabe mencionar que para este tipo de trazado solo se arroja la primera intersección que se encuentra, es decir, sólo se continúa probando en alguna posición del arreglo de colores mientras el color actual en esa posición sea la de fondo.

El número de puntos de control así como una variable que permite cambiar el número de cambios en el punto de control se mencionan con mayor detalle en el apéndice C de este documento.

A continuación un diagrama que describe de manera general el proceso para generar una imagen con el algoritmo de trazado de rayos modificado de nuestra aplicación.

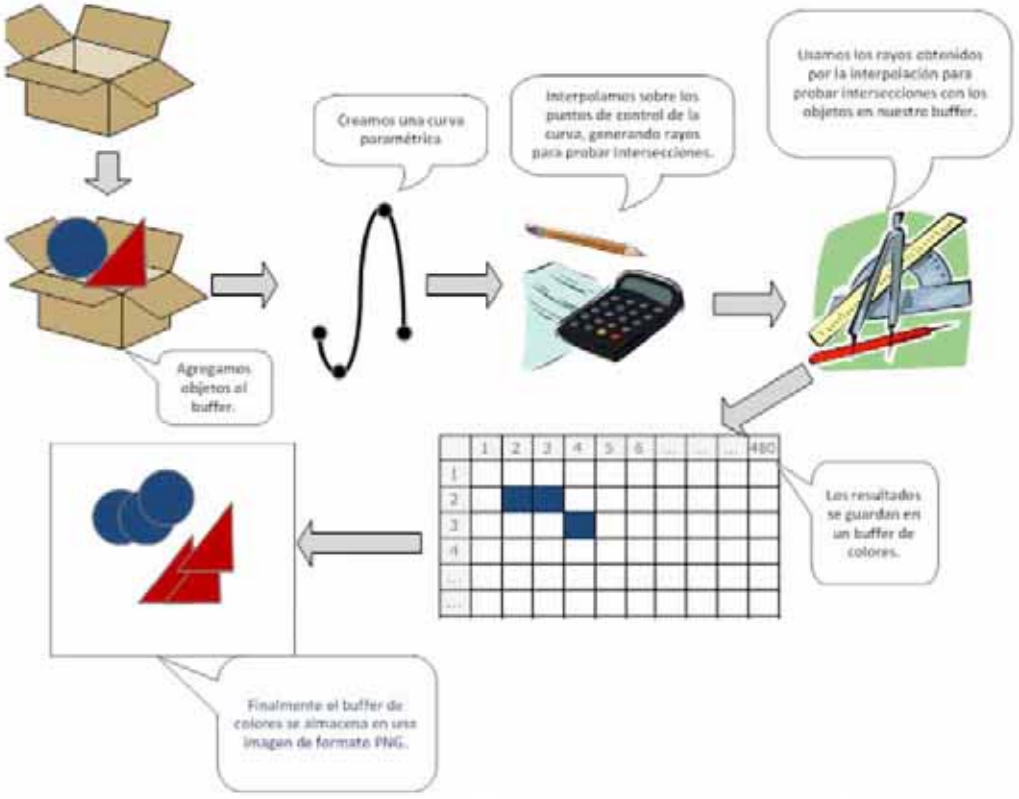

Ilustración 28. Diagrama general del algoritmo de trazado de rayos modificado.

C. Referencia de uso

Esta referencia va dirigida a los programadores que deseen modificar el código fuente para crear imágenes modificando los parámetros que a continuación se describen, así como agregar nuevas funcionalidades para enriquecer el proyecto, es decir, no es un manual para usuario final.

Partes principales de la aplicación.

La aplicación cuenta con dos partes esenciales:

- Main. Es la clase principal y donde se encuentra el buffer de objetos. En esta clase se agregan las primitivas de trazado que se desea aparezcan en la imagen. La ruta de acceso a esta clase es: /Ray_Tracing_vlei/main.cpp.
- RayTracer. En esta clase se encuentran las funciones de trazado, tanto la de trazado normal como la del algoritmo de trazado de rayos modificado. Más adelante se describe su funcionamiento y la modificación de los parámetros principales para crear algunos efectos. La ruta de acceso a esta clase es la siguiente: /Ray_Tracing_vlei/RayTracer.cpp.

¿Cómo agregar objetos al buffer?

En la clase main después de la línea donde se declara el vector ó buffer de objetos llamado *objList* se comienzan a agregar las primitivas. La ilustración siguiente muestra las líneas despúes de las cuales podemos comenzar a llenar el buffer de objetos.

```
int main()
{
   cout << "C++ Ray Tracing..." << endl;
   cout << "Created by Allen Sherrod..." << endl ;
   cout << "Modified by Edgar Vazquez..." << endl<< endl;
 RayTracer rayTracer(600, 480, -255);
   vector<Shape*> objList;
```

Ilustración 29. Después de estas líneas agregamos los objetos.

Cómo podemos observar nuestra variable de buffer ó vector de objetos es llamada objList y haremos uso de ella para agregar las primitivas.

Primitivas de trazado implementadas

A continuación se describen las primitivas con las que cuenta la aplicación actualmente y su manera de agregarse al buffer de objetos. Los puntos en el espacio se manejan como variables tipo *Vector3D* el cual recibe tres parámetros (x,y,z); los colores se manejan con formato RGB con variables de tipo *Color3*.

Esfera

Cuenta con un origen, un radio y un color. El origen es el punto dónde se ubica la esfera en el espacio, este valor junto con el radio son utilizados para el cálculo de la intersección, en caso de haberla se regresa el color de la esfera como resultado. Una esfera se agrega al buffer de objetos de la siguiente manera.

```
// Create the apple red sphere.
objList.push_back(new Sphere(Vector3D(65, 173, -550),
                  20, Color3(238.0.0)));
```

Ilustración 30. Agregar una esfera al buffer de objetos.

La imagen anterior se interpreta de la siguiente manera:
Origen: (65, 173, -550)
Radio: 20
Color:

Triángulo

Cuenta con 3 puntos en el espacio los cuales trazan el triángulo entre ellos. Además de un color para regresar si algún rayo lo golpea en su superficie. Un triángulo se agrega de la siguiente manera.

```
/*Indigo*/
  objList.push_back(new Triangle(Vector3D( -2800, -1500, 900),
                                 Vector3D( -2800, 200, 900),
                                 Vector3D( -3100, 200, 900),
                                 Color3(75, 0, 130)));
```

Ilustración 31. Agregar un triángulo al buffer de objetos.

La ilustración anterior se interpreta de la siguiente forma:
Punto A: (-2800, -1500, 900)
Punto B: (-2800, 200, 900)
Punto C: (-3100, 200, 900)
Color:

Cilindro

Cuenta con un punto en el espacio como centro, un vector como orientación, así como radio, altura y color. Un cilindro se agrega de la siguiente manera.

```
objList.push_back(new Cylinder(Vector3D(-200,100,-100),Vector3D(1,0,0),20,10,
    Color3(0,255,255)));
```

Ilustración 32. Agregar un cilindro al buffer de objetos.

Lo mostrado en la ilustración se interpreta así:
Centro: (-200, 100, -100)
Orientación: (1, 0, 0)
Radio: 20
Altura: 10
Color:

¿Cómo se traza una imagen de manera normal?

Para cambiar la forma en que se genera una imagen vamos a la clase RayTracer.cpp y buscamos la siguiente sección de código.

```
// Normal_Trace(){ se comenta
  for( y = 0; y < height; y++)
   {
      for(x = 0; x < width; x++)
         {
                primaryRay.direction = Vector3D(x - (float)(width >> 1), y - (float)(height >> 1),depth);
                primaryRay.direction.Normalize();
                primaryBuffer[index] = Trace(primaryRay, objList);
                index++;
         }
   }/**/
 //}hasta aqui se comenta
```

Ilustración 33. Código para trazar normalmente

Cómo tenemos las dos funciones en la misma clase debemos comentar la del algoritmo modificado como se muestra en la siguiente ilustración.

Ilustración 34. Función del algoritmo modificado comentada.

Cabe mencionar que la variable más importante usando este tipo de trazado es la variable *primaryRay.origin* que nos permite modificar el inicio de los rayos que se trazarán para posteriormente comprobar intersecciones con los objetos. Lo anterior conlleva que dependiendo de nuestro punto de inicio se mostrarán o no y de cierta u otra manera los objetos en el buffer.

```
primaryRay.origin = Vector3D( 50, -75, -250);
```

Ilustración 35. Punto de inicio de los rayos.

¿Cómo generar imágenes con el algoritmo de trazado de rayos modificado?

Primero que nada debemos comentar nuestra función de trazado normal para que quede como se muestra en la ilustración siguiente.

```
// Normal_Trace(){ se comenta
  /*for( y = 0; y < height; y++)
    {
       for(x = 0; x < width; x++)
          {
                  primaryRay.direction = Vector3D(x - (float)(width >> 1), y - (float)(height >> 1),depth);
                  primaryRay.direction.Normalize();
                  primaryBuffer[index] = Trace(primaryRay, objList);
                  index++;
          }
    }/**/
  //hasta aqui se comenta
```

Ilustración 36. Función de trazado normal comentada

Ahora quitamos el comentario de nuestra función de trazado de rayos modificado, la figura siguiente ilustra cómo debería lucir ese fragmento de código.

Ilustración 37. Función de trazado de rayos modificada activa.

Ahora las variables importantes en esta función son las siguientes:

```
primaryRay.origin = Vector3D( -150, -100, -200);
```

Ilustración 38. Punto de origen de los rayos

El punto de la ilustración anterior funge una función distinta que el descrito en el trazado normal, en este caso funciona como primer punto de control de la curva paramétrica que crearemos a continuación.

```
MgcBezierCurve3 pBezier(3,valoresPrueba);

pBezier.m_akCtrlPoint[1].x =  550;
pBezier.m_akCtrlPoint[1].y = -200;
pBezier.m_akCtrlPoint[1].z = -250;

pBezier.m_akCtrlPoint[2].x = -550;
pBezier.m_akCtrlPoint[2].y = -200;
pBezier.m_akCtrlPoint[2].z = -650;

pBezier.m_akCtrlPoint[3].x = -450;
pBezier.m_akCtrlPoint[3].y = -100;
pBezier.m_akCtrlPoint[3].z = -650;
```

Ilustración 39. Creación de la curva paramétrica.

La ilustración anterior muestra la creación de otra variable importante en esta función. La curva paramétrica cuenta con 3 puntos de control, los cuales ser interpolarán para generar segmentos de línea que utilizaremos como rayos para hacer las pruebas de intersección con los objetos en el buffer.

```
int iCP = 0;//indice de los puntos de control
```

Ilustración 40. Punto de control de inicio.

La ilustración 40 muestra la definición de una variable que controla el punto de control con el cual se iniciará la interpolación que describiremos más adelante.

```
float fPeso = 0.0f;//para la interpolacion de los puntos de control
```

Ilustración 41. Variable para controlar la interpolación.

La variable fPeso controla el incremento que se tiene para la interpolación, a medida que disminuye se generan por consiguiente más segmentos de línea entre los puntos de control, es decir, más rayos para probar la intersección con los objetos.

```
increment = 480;
```

Ilustración 42. Cambio en el punto de control.

La variable increment modifica el número de píxeles que se trazan con un mismo punto de control en este caso se traza toda la imagen con un mismo punto de control, si su valor fuera por ejemplo de 240 se trazaría con 2 cambios en el punto de control, si fuese 120 los cambios serían 4 y así sucesivamente. Esto provoca efectos como los mostrados en la ilustración 12.

```
for( fPeso=0.0f; fPeso<1.0f;fPeso+=0.1f)
{
        rayoInterpolado = (pBezier.m_akCtrlPoint[iCP]*fPeso)+(pBezier.m_akCtrlPoint[iCP+1]*(1-fPeso))/2;
```

Ilustración 43. Interpolación

La imagen 43 muestra las líneas encargadas de la interpolación, como podemos observar se realiza con dos puntos de control y los rayos se generan cada vez dependiendo del peso asignado a cada uno. El incremento es lo que determina la cantidad de rayos que se generan. Se debe tener cuidado de no darle un valor muy pequeño ya que el tiempo que la aplicación tardará en renderizar la imagen puede aumentar en varios minutos.

Sea cuál sea la función con la que se vaya a generar la imagen lo que sigue después de las modificaciones anteriores es compilar y ejecutar la clase main.

¿Dónde se guardan las imágenes?

Después de ejecutar la clase main la imagen generada se encuentra en la ruta /RayTracing_vlei/WinVS2005/RayTracedScene.png.

Dentro de la carpeta /Images_History se incluyeron imágenes generadas a lo largo del desarrollo del proyecto, en ellas se pueden observar muchos efectos generados de la manera descrita en este ápendice.

El proyecto fue creado usando Visual Studio 2008 Express Edition por lo que se recomienda abrir el archivo RayTracing.VC++Project ubicado en /RayTracing_vlei/WinVS2005/.