UNIDAD I

Introducción a la Graficación por Computadora

Materia: Graficación | Blender & Python

1.1 Historia y Evolución de la Graficación por Computadora

La graficación por computadora es un área de la informática que se ocupa de la generación, manipulación y representación visual de datos mediante computadoras. Su historia es inseparable del desarrollo del hardware y el software desde mediados del siglo XX.

Década de 1950 – Los Orígenes

Los primeros indicios de graficación digital surgieron con el MIT Whirlwind (1951), una de las primeras computadoras capaz de mostrar gráficos en tiempo real sobre una pantalla de tubo de rayos catódicos (CRT). En 1958, la compañía Calcomp desarrolló los primeros plotters automáticos, dispositivos que trazaban diagramas a partir de señales digitales.

Década de 1960 – El Nacimiento Formal

En 1962, Ivan Sutherland presentó su tesis doctoral en el MIT bajo el nombre Sketchpad: A Man-Machine Graphical Communication System. Este trabajo es considerado el punto de partida formal de la graficación interactiva, ya que permitía dibujar directamente sobre una pantalla usando una pluma luminosa. Sutherland introdujo conceptos fundamentales como transformaciones geométricas, instancias de objetos y restricciones.

También en esta década, la empresa IBM comenzó a integrar terminales gráficas en sus equipos, y la NASA utilizó graficación computacional para la simulación de trayectorias espaciales.

Década de 1970 – Modelado y Rasterización

En los años 70 surgieron los primeros algoritmos de rasterización eficientes. Henri Gouraud (1971) propuso el sombreado suave de superficies que lleva su nombre. Bui Tuong Phong (1975) desarrolló el modelo de iluminación especular que es estándar hasta hoy. Además, Ed Catmull (cofundador de Pixar) introdujo el mapeo de texturas y la subdivisión de superficies.

En 1972, la película Westworld utilizó por primera vez animación digital generada por computadora en el cine.

Década de 1980 – Democratización y Estándares

La aparición de las computadoras personales (Apple II, IBM PC) y las primeras tarjetas gráficas dedicadas hizo accesible la graficación a un público más amplio. En 1982, Tron fue el primer largometraje con secuencias CGI extensas. En 1984, Pixar lanzó The Adventures of André and Wally B., y en 1986 se fundó como estudio independiente. También en esta época surgieron los primeros estándares gráficos como GKS (Graphical Kernel System) y PHIGS.

Décadas de 1990-2000 – Aceleración por Hardware y Tiempo Real

La aparición de las GPU (Graphics Processing Units) dedicadas, como la NVIDIA GeForce 256 en 1999, marcó un hito al realizar operaciones de transformación e iluminación completamente en hardware. Las APIs OpenGL y DirectX se convirtieron en estándares de la industria. Películas como Jurassic Park (1993) y Toy Story (1995, primer largometraje 100% CGI) mostraron el potencial de la graficación tridimensional.

Siglo XXI – Realismo, Tiempo Real e Inteligencia Artificial

Hoy en día, los motores de gráficos como Unreal Engine y Unity permiten visualizaciones en tiempo real con niveles de realismo fotográfico. La técnica de ray tracing en tiempo real (implementada en las GPU RTX de NVIDIA desde 2018) calcula el comportamiento físico de la luz en cada fotograma. La inteligencia artificial generativa ha abierto nuevas fronteras con herramientas como DALL-E, Stable Diffusion y Midjourney. Blender, el software open-source tridimensional, se ha consolidado como una herramienta profesional usada en cine, videojuegos y visualización científica.

1.2 Áreas de Aplicación

La graficación computacional permea prácticamente todos los sectores productivos y de investigación modernos. A continuación se describen las áreas principales:

Entretenimiento y Medios Digitales

Es quizá el área más visible. Los efectos visuales (VFX) en cine y televisión dependen completamente de la graficación 3D. Películas como Avatar, Avengers: Endgame y cualquier producción de animación moderna son ejemplos directos. Los videojuegos representan la mayor industria de entretenimiento global, y la graficación en tiempo real es su núcleo tecnológico.

Diseño Asistido por Computadora (CAD/CAM)

Ingenieros y arquitectos utilizan software como AutoCAD, SolidWorks o Catia para diseñar piezas, edificios y sistemas complejos antes de fabricarlos. La graficación permite detectar errores de diseño antes de la producción física, ahorrando enormes costos.

Medicina y Ciencias de la Salud

La visualización médica es crítica: tomografías computarizadas (CT), resonancias magnéticas (MRI) y ultrasonidos generan imágenes en 2D y 3D del interior del cuerpo humano. La cirugía asistida por computadora y la planificación quirúrgica en realidad aumentada son aplicaciones emergentes de alto impacto.

Simulación Científica

La dinámica de fluidos computacional (CFD), la simulación de fenómenos climáticos, la visualización de datos astronómicos y la representación molecular son áreas donde la graficación permite comprender fenómenos imposibles de observar directamente.

Educación y Capacitación

Los simuladores de vuelo, los entornos virtuales de entrenamiento médico o militar, y las plataformas de e-learning con visualizaciones interactivas son ejemplos de cómo la graficación transforma el aprendizaje. En este contexto, Blender es una herramienta pedagógica de gran valor por ser gratuita y de código abierto.

Realidad Virtual y Aumentada (VR/AR)

Dispositivos como el Meta Quest o las Apple Vision Pro requieren renderizado estereoscópico a 90 fps o más para evitar el mareo por movimiento. La AR superpone gráficos sobre el mundo real en aplicaciones industriales, de mantenimiento, educativas y de entretenimiento.

Visualización de Datos

El Big Data y la ciencia de datos dependen de visualizaciones para comunicar patrones y tendencias complejas. Herramientas como Tableau, D3.js o matplotlib transforman datos numéricos en representaciones gráficas comprensibles.

1.3 Aspectos Matemáticos de la Graficación

La graficación computacional descansa sobre fundamentos matemáticos sólidos. Comprender estos conceptos es esencial para desarrollar cualquier aplicación gráfica, desde el trazado de una simple línea hasta la renderización de escenas tridimensionales complejas.

Sistemas de Coordenadas

Un sistema de coordenadas cartesianas 2D define cada punto mediante un par (x, y). En 3D se extiende a una terna (x, y, z). Blender utiliza un sistema dextrógiro donde el eje Z apunta hacia arriba. La comprensión de los sistemas de coordenadas locales versus globales es fundamental para el modelado y la animación.

Geometría Vectorial

Los vectores son la base de la representación de posición, dirección y magnitud en graficación. Operaciones clave incluyen la suma y resta vectorial, el producto punto (dot product) para calcular ángulos entre vectores, y el producto cruzado (cross product) para obtener normales de superficies. Las normales son vectores perpendiculares a una superficie y son esenciales para el cálculo de iluminación.

Álgebra Matricial y Transformaciones

Las transformaciones geométricas se representan mediante matrices. Las tres transformaciones fundamentales son la traslación (T), la rotación (R) y el escalado (S). En graficación 3D se utilizan matrices 4×4 con coordenadas homogéneas, lo que permite combinar todas las transformaciones en una sola operación multiplicativa:

$$M = T \cdot R \cdot S$$

Las coordenadas homogéneas representan un punto 3D (x, y, z) como (x, y, z, 1) y un vector como (x, y, z, 0). Esta representación es fundamental en OpenGL y en el pipeline de renderizado de Blender (EEVEE/Cycles).

Trigonometría Aplicada

La trigonometría es omnipresente en graficación. Las funciones seno y coseno se utilizan para calcular posiciones en círculos y esferas, para rotaciones, para ondas de animación y para efectos procedurales. En el script de la Flor de la Vida (sección 1.5.1) se puede observar su aplicación directa:

$$x = r \cdot \cos(\theta) \qquad y = r \cdot \sin(\theta)$$

Estas ecuaciones convierten coordenadas polares (radio, ángulo) a coordenadas cartesianas (x, y), lo que permite posicionar objetos en patrones circulares con precisión matemática.

Interpolación y Curvas

La interpolación lineal (Lerp) y las curvas de Bézier son esenciales para animación suave y modelado de formas orgánicas. Una curva de Bézier cúbica se define por cuatro puntos de control y su ecuación paramétrica es:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$

Blender utiliza curvas de Bézier y NURBS tanto para modelado como para rutas de animación (F-Curves).

Proyección y Perspectiva

La proyección transforma el espacio 3D al plano 2D de la pantalla. Existen dos tipos principales: la proyección ortogonal, que mantiene proporciones reales y se usa en planos técnicos, y la proyección en perspectiva, que simula la visión humana y es la más utilizada en gráficos interactivos. La matriz de

proyección en perspectiva en OpenGL involucra el campo de visión (FOV), la relación de aspecto y los planos near/far.

1.4 Modelos del Color: RGB, CMY, HSV y HSL

El color es un elemento central de la graficación. Existen diferentes modelos matemáticos para representarlo, cada uno optimizado para un propósito específico: visualización en pantalla, impresión, o selección intuitiva por parte del artista.

Modelo RGB (Red, Green, Blue)

Es el modelo aditivo estándar para pantallas y dispositivos de visualización. Se basa en la mezcla de luz de tres colores primarios: rojo (R), verde (G) y azul (B). Cada componente varía de 0 a 255 (8 bits) o de 0.0 a 1.0 (normalizado). La combinación a máxima intensidad de los tres produce blanco puro (255, 255, 255), y la ausencia de todos produce negro (0, 0, 0).

En Blender, todos los materiales y nodos de color trabajan internamente con valores RGB normalizados (0.0–1.0) en espacio lineal para mayor precisión en los cálculos de iluminación.

Modelo CMY / CMYK (Cyan, Magenta, Yellow, Key)

Es el modelo sustractivo utilizado en impresión. En lugar de emitir luz, las tintas absorben longitudes de onda del espectro visible. El cián absorbe el rojo, el magenta absorbe el verde y el amarillo absorbe el azul. La conversión desde RGB es: C = 1 - R, M = 1 - G, Y = 1 - B. En la práctica industrial se agrega el componente K (negro puro) por eficiencia económica y para obtener negros más profundos.

Modelo HSV (Hue, Saturation, Value)

El modelo HSV fue desarrollado por Alvy Ray Smith en 1978 para hacer la selección de color más intuitiva. Sus tres componentes son:

- Hue (Matiz): El tipo de color, representado como un ángulo de 0° a 360° en una rueda de color. 0° es rojo, 120° es verde, 240° es azul.
- Saturation (Saturación): La pureza del color, de 0 (gris) a 1 (color puro).
- Value (Valor/Brillo): La luminosidad del color, de 0 (negro) a 1 (máximo brillo).

Blender incluye el selector HSV en su panel de color, muy utilizado por artistas para ajustar tonos sin afectar otros canales.

Modelo HSL (Hue, Saturation, Lightness)

Similar al HSV pero con la componente de Lightness (luminosidad) en lugar de Value. La diferencia clave es que en HSL, un valor de Lightness = 1.0 siempre produce blanco puro, independientemente del Hue y la Saturation. En HSV, Value = 1.0 produce el color puro si la saturación es máxima. HSL se utiliza ampliamente en CSS/web y en ajustes de imagen.

Tutorial Práctico: Iluminación de un Cubo en Blender

A continuación se describe el procedimiento para crear un cubo con material de color personalizado e iluminarlo correctamente en Blender 3.x o superior, demostrando los modelos de color en la práctica.

Paso 1: Configurar la Escena

Abrir Blender y asegurarse de estar en el espacio de trabajo General. Eliminar el cubo predeterminado si existe (Tecla X → Delete). En el menú Add → Mesh → Cube, agregar un nuevo cubo. Posicionarlo en el origen (G, Z, 0, Enter para bajarlo al piso si se desea).

Paso 2: Agregar una Fuente de Luz

Ir a Add → Light → Point Light. Posicionar la luz en (4, -4, 6) usando el panel N → Item → Location. En el panel de propiedades (ícono de bombilla), aumentar la potencia a 1000W y cambiar el color haciendo clic en el selector de color. Aquí se puede experimentar con el modelo RGB ajustando los canales o cambiando al modo HSV en el selector.

Paso 3: Agregar una Luz de Ambiente (HDRI)

En las propiedades del mundo (ícono de esfera), activar el nodo de fondo. Usar un color de ambiente o conectar una textura HDRI para iluminación global realista. Esto simula la iluminación de entorno mediante Image-Based Lighting (IBL).

Paso 4: Asignar Material al Cubo

Con el cubo seleccionado, ir a Propiedades de Material (ícono de esfera con brillo). Hacer clic en New para crear un nuevo material. En el campo Base Color, hacer clic para abrir el selector de color. Explorar los diferentes modos: RGB (valores numéricos precisos), HSV (ajuste artístico intuitivo) y Hex (código hexadecimal para compatibilidad web). Ajustar Roughness (rugosidad) a 0.3 para un acabado semiglossy. Ajustar Metallic a 0.0 para un material dieléctrico no metálico.

Paso 5: Iluminar Caras Individualmente con Vertex Paint

Para colorear caras individuales del cubo, entrar en Edit Mode (Tab), seleccionar una cara con A para deseleccionar y luego clic en la cara deseada. En el panel de materiales, crear un nuevo slot de material, asignar un color diferente y presionar Assign. Esto permite que cada cara del cubo tenga su propio material con modelo de color independiente, demostrando la aplicación práctica de RGB, HSV y HSL en una misma escena.

Paso 6: Renderizar

Cambiar el motor de render a EEVEE para visualización rápida o Cycles para ray tracing físicamente correcto. Presionar F12 para renderizar. Observar cómo la iluminación interactúa con los materiales y el modelo de color aplicado.

1.5 Representación y Trazo de Líneas y Polígonos

La representación de primitivas geométricas como líneas y polígonos es la base de la graficación vectorial y del modelado 3D. Comprender cómo se generan matemáticamente y cómo se trazan sobre una pantalla es fundamental para cualquier ingeniero en sistemas computacionales especializado en gráficos.

Representación de Líneas

Una línea recta en un plano 2D se representa matemáticamente mediante la ecuación: y = mx + b, donde m es la pendiente y b es el intercepto en y. Sin embargo, en graficación rasterizada, esta representación continua debe convertirse en píxeles discretos. Los algoritmos más importantes para el trazado de líneas son:

Algoritmo DDA (Digital Differential Analyzer)

El DDA calcula la pendiente m = (y2-y1)/(x2-x1) e incrementa x por 1 mientras calcula y = y + m, redondeando al entero más cercano en cada paso. Es simple pero usa aritmética de punto flotante, lo que lo hace relativamente lento.

Algoritmo de Bresenham

Desarrollado por Jack Bresenham en 1962, este algoritmo usa exclusivamente aritmética entera, haciéndolo muy eficiente para hardware. Utiliza un parámetro de decisión pk para determinar en cada paso qué píxel es más cercano a la línea matemática ideal. Es el algoritmo estándar implementado en la mayoría del hardware gráfico.

Representación de Polígonos

Un polígono regular de n lados puede representarse mediante sus vértices, calculados con coordenadas polares:

$$v_i = (r \cdot \cos(2\pi i/n),\ r \cdot \sin(2\pi i/n)) \quad \text{para } i = 0, 1, \ldots, n-1$$

En graficación 3D, los polígonos (especialmente los triángulos) son la primitiva fundamental de renderizado. Blender almacena la malla como una estructura de datos que contiene listas de vértices, aristas y caras (BMesh), lo que corresponde a la representación Half-Edge o Winged-Edge en la teoría de geometría computacional.

Pipeline de Rasterización

El proceso de convertir primitivas geométricas a píxeles en pantalla sigue el pipeline gráfico: primero las transformaciones de vértices (Model → View → Projection), luego el clipping (recorte de lo que queda fuera del frustum), la rasterización (conversión a fragmentos/píxeles), el procesamiento de fragmentos (aplicación de texturas y shaders) y finalmente el blending para transparencias.

1.5.1 Formatos de Imagen

Los formatos de imagen digital determinan cómo se almacena, comprime y recupera la información visual. Elegir el formato correcto es crítico para optimizar calidad, tamaño de archivo y compatibilidad.

PNG (Portable Network Graphics)

Formato sin pérdida con soporte para canal alfa (transparencia). Utiliza compresión DEFLATE. Ideal para gráficos con bordes nítidos, interfaces de usuario, texturas con transparencia y capturas de pantalla. No es adecuado para fotografías de alta resolución donde el tamaño de archivo puede ser excesivo.

JPEG (Joint Photographic Experts Group)

Formato con pérdida optimizado para fotografías. Utiliza la Transformada Discreta del Coseno (DCT) para comprimir eliminando información visual de alta frecuencia que el ojo humano no percibe bien. El nivel de compresión es ajustable (calidad 0-100). No soporta transparencia. Inadecuado para imágenes con texto o bordes nítidos.

EXR (OpenEXR)

Formato HDR (High Dynamic Range) creado por Industrial Light & Magic. Soporta 16 o 32 bits por canal en punto flotante, permitiendo almacenar valores de luminancia más allá del rango 0-1. Es el formato estándar en VFX y producción cinematográfica. Blender lo usa como formato principal para renders compositing.

TIFF (Tagged Image File Format)

Formato sin pérdida de alta calidad, muy usado en fotografía profesional e impresión. Soporta múltiples capas, canales alfa y profundidades de color de 8, 16 o 32 bits. Su principal desventaja es el gran tamaño de archivo.

SVG (Scalable Vector Graphics)

A diferencia de los anteriores (formatos raster), SVG es un formato vectorial basado en XML. Las imágenes SVG se describen matemáticamente mediante primitivas geométricas y pueden escalarse a cualquier tamaño sin pérdida de calidad. Ampliamente utilizado en iconografía web, logotipos y visualizaciones de datos.

Comparativa Resumida

PNG: sin pérdida, transparencia, web. | JPEG: con pérdida, fotos, web. | EXR: HDR, VFX, renders. | TIFF: sin pérdida, impresión. | SVG: vectorial, escalable, web.

Práctica 1: Dibujo de un Polígono Regular en Blender

El siguiente script en Python para Blender genera un polígono regular de n lados mediante cálculo matemático de coordenadas cartesianas a partir de coordenadas polares. El script demuestra los conceptos teóricos de la sección 1.3 (trigonometría) y 1.5 (representación de polígonos) en una implementación práctica y funcional.

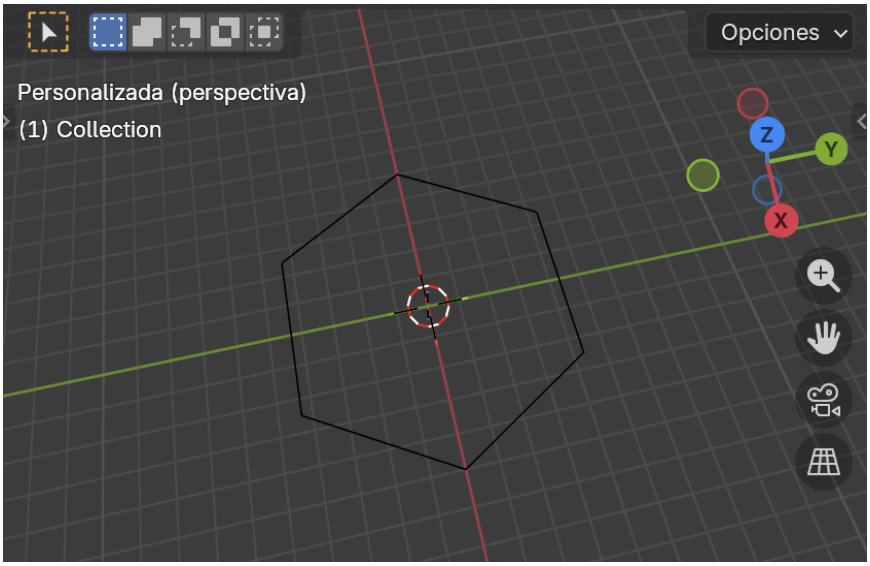
Descripción del Algoritmo

El script utiliza la API bpy de Blender para crear una malla vacía y poblarla con vértices calculados matemáticamente. Para un polígono de n lados con radio r, el vértice i-ésimo se posiciona en el ángulo $\theta_i = 2\pi i/n$. Las aristas conectan vértices consecutivos mediante el operador módulo (%) para cerrar el polígono.

Código: Polígono Regular

Copiar el siguiente código en el Script Editor de Blender (Scripting workspace) y ejecutar con Run Script o Alt+P:

```
import bpy
import math

def crear_poligono_2d(nombre, lados, radio):
    """
    Crea un polígono regular plano en el origen (0,0,0).
    Parámetros:
      - nombre: Nombre del objeto en Blender.
      - lados: Cantidad de vértices (ej. 6 para hexágono).
      - radio: Distancia del centro a los vértices.
    """
    malla = bpy.data.meshes.new(nombre)
    objeto = bpy.data.objects.new(nombre, malla)
    bpy.context.collection.objects.link(objeto)
    vertices = []
    aristas = []

    for i in range(lados):
        angulo = 2 * math.pi * i / lados
        x = radio * math.cos(angulo)
        y = radio * math.sin(angulo)
        vertices.append((x, y, 0))

    for i in range(lados):
        aristas.append((i, (i + 1) % lados))

    malla.from_pydata(vertices, aristas, [])
    malla.update()

# LIMPIEZA DE ESCENA
bpy.ops.object.select_all(action='SELECT')
bpy.ops.object.delete()

# LLAMADA A LA FUNCIÓN
crear_poligono_2d("Poligono2D", lados=6, radio=5)
```



Parámetros Configurables

- nombre: Cadena de texto para identificar el objeto en la escena de Blender.
- lados: Número entero de vértices. Usar 3 para triángulo, 4 para cuadrado, 5 para pentágono, 6 para hexágono, etc.
- radio: Valor flotante que determina el tamaño del polígono en unidades de Blender.

Análisis del Código

La función crear_poligono_2d encapsula toda la lógica de creación. Primero crea la estructura de datos de la malla (bpy.data.meshes.new) y el objeto contenedor (bpy.data.objects.new). El objeto debe vincularse explícitamente a la colección de la escena con objects.link para ser visible. El bucle for calcula cada vértice mediante las ecuaciones trigonométricas y los almacena en una lista. Un segundo bucle define las aristas como pares de índices. Finalmente, from_pydata construye la geometría y malla.update() valida los cambios.

Práctica 2: La Flor de la Vida

La Flor de la Vida es un patrón geométrico sagrado compuesto por múltiples círculos que se superponen en una disposición hexagonal. Aparece en culturas de todo el mundo y es un ejemplo perfecto de cómo los patrones matemáticos generan estructuras de gran belleza visual. En este script se genera computacionalmente usando conceptos de coordenadas polares y bucles while.

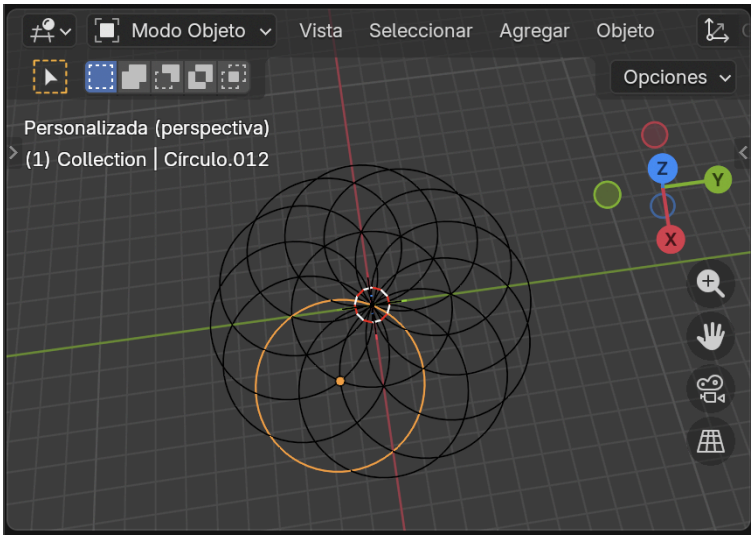
Fundamento Matemático

El patrón consiste en un círculo central rodeado por círculos de igual radio cuyos centros están ubicados a una distancia igual al radio del círculo central (o un múltiplo de él). Con paso_angular = 30°, se generan 12 círculos en la primera capa, creando el patrón base. Con paso_angular = 60° se obtienen 6 círculos para el Fruto de la Vida.

Código: Flor de la Vida

Ejecutar en el Script Editor de Blender:

```
import bpy
import math

# Limpiar escena
bpy.ops.object.select_all(action='SELECT')
bpy.ops.object.delete()

# Parámetros de la figura
radio = 3
angulo_actual = 0
paso_angular = 30  # Cada 30 grados para 12 círculos

# 1. Círculo Central
bpy.ops.mesh.primitive_circle_add(radius=radio, location=(0, 0, 0), vertices=64)

# --- PATRÓN REPETITIVO CON WHILE ---
while angulo_actual < 360:
    x = radio * math.cos(math.radians(angulo_actual))
    y = radio * math.sin(math.radians(angulo_actual))
    bpy.ops.mesh.primitive_circle_add(radius=radio, location=(x, y, 0), vertices=64)
    angulo_actual += paso_angular
```



Análisis del Código

El script comienza limpiando la escena con select_all y delete, lo que garantiza un entorno de trabajo limpio. Luego crea el círculo central en el origen. El bucle while itera desde angulo_actual = 0 hasta angulo_actual < 360, incrementando en paso_angular grados por iteración. En cada iteración, las ecuaciones x = radio·cos(θ) e y = radio·sin(θ) calculan el centro del siguiente círculo. La función primitive_circle_add crea el círculo y vertices=64 define la suavidad de la curva.

Este ejercicio demuestra la potencia de combinar matemáticas básicas (trigonometría, coordenadas polares) con un bucle iterativo para generar patrones complejos a partir de reglas simples, un principio fundamental tanto en graficación computacional como en el diseño generativo y procedural.

1.6 Procesamiento de Mapas de Bits

Los mapas de bits (bitmaps o imágenes raster) son representaciones digitales de imágenes donde cada píxel almacena información de color de forma independiente. El procesamiento de estas imágenes abarca una amplia gama de técnicas que van desde ajustes básicos hasta transformaciones complejas mediante algoritmos matemáticos.

Estructura de un Mapa de Bits

Una imagen raster se puede entender como una matriz bidimensional de píxeles de dimensiones W × H (ancho × alto). Cada píxel es un vector de valores numéricos que representan los canales de color. En una imagen RGB de 8 bits, cada canal tiene valores de 0 a 255, por lo que existen $256^3 \approx 16.7$ millones de colores posibles. Con un canal alfa adicional (RGBA), cada píxel ocupa 4 bytes.

Operaciones de Punto

Las operaciones de punto modifican cada píxel de forma independiente. Los ajustes de brillo (sumar una constante a todos los canales), contraste (escalar los valores respecto al punto medio), gamma (aplicar una función de potencia) e inversión (valor_nuevo = 255 - valor_original) son ejemplos típicos. En Blender, estas operaciones se realizan en el Compositor usando nodos como Bright/Contrast, Gamma, Invert y Curves.

Operaciones de Convolución y Filtros

Los filtros de convolución aplican una máscara (kernel) deslizante sobre la imagen. El valor de cada píxel de salida se calcula como la suma ponderada de sus vecinos en la imagen de entrada. Los filtros de desenfoque (blur) usan kernels de promedios o gaussianos que suavizan la imagen eliminando ruido de alta frecuencia. Los filtros de realce de bordes (Sobel, Laplaciano) detectan discontinuidades en la imagen calculando gradientes. El filtro Sharpen aumenta el contraste local, haciendo la imagen más nítida.

Transformaciones Morfológicas

La erosión y la dilatación son operaciones morfológicas que modifican la forma de regiones binarias en una imagen. La erosión reduce las regiones blancas (útil para eliminar ruido pequeño) y la dilatación las expande (útil para cerrar huecos). La combinación de ambas produce operaciones de apertura y cierre muy útiles en visión por computadora.

Histograma y Ecualización

El histograma de una imagen es la distribución de frecuencias de los valores de cada canal de color. La ecualización del histograma redistribuye estos valores para maximizar el contraste global de la imagen. Es una técnica estándar en procesamiento de imágenes médicas y satelitales. En Blender, el nodo Levels del Compositor permite ajustar los puntos negros, medios y blancos del histograma.

Texturas y Mapeo UV en Blender

El mapeo UV es el proceso de proyectar una imagen 2D (mapa de textura) sobre la superficie de un objeto 3D. El proceso requiere desempaquetar la malla (UV Unwrap) para crear una representación plana de su superficie, luego asignar una imagen a las coordenadas UV. Blender ofrece múltiples métodos de desempaquetado: Smart UV Project, Follow Active Quads, Unwrap (basado en costuras) y proyecciones planares, cilíndricas o esféricas.

Los tipos de mapas de bits más comunes en materiales PBR (Physically Based Rendering) son: Albedo/Diffuse (color base), Normal Map (detalle de superficie), Roughness (rugosidad), Metallic (reflectividad metálica), Ambient Occlusion (sombras de contacto) y Displacement (geometría adicional).

Compresión de Imágenes

Los algoritmos de compresión reducen el tamaño del archivo de una imagen. La compresión sin pérdida (PNG, GIF, TIFF con LZW) recupera exactamente los datos originales al descomprimir. La compresión con pérdida (JPEG, WebP en modo lossy) descarta información visual de baja importancia perceptual. Los algoritmos de compresión para GPU (DXT/BC, ASTC, ETC) permiten mantener texturas comprimidas directamente en la memoria de la GPU, reduciendo el ancho de banda de memoria.

Bibliografía

Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1990). *Computer Graphics: Principles and Practice* (2nd ed.). Addison-Wesley.

Hill, F. S., & Kelley, S. M. (2006). *Computer Graphics Using OpenGL* (3rd ed.). Pearson Prentice Hall.

Hearn, D., Baker, M. P., & Carithers, W. (2011). *Computer Graphics with OpenGL* (4th ed.). Pearson.

Shirley, P., & Marschner, S. (2009). *Fundamentals of Computer Graphics* (3rd ed.). A K Peters/CRC Press.

Sutherland, I. E. (1963). *Sketchpad: A man-machine graphical communication system* [Tesis doctoral, MIT]. Massachusetts Institute of Technology.

Blender Foundation. (2024). *Blender Reference Manual (Version 4.x)*. https://docs.blender.org/manual/en/latest/

Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed.). Pearson.

Smith, A. R. (1978). *Color gamut transform pairs*. ACM SIGGRAPH Computer Graphics, 12(3), 12–19. https://doi.org/10.1145/965139.807361

Bresenham, J. E. (1965). *Algorithm for computer control of a digital plotter*. IBM Systems Journal, 4(1), 25–30. https://doi.org/10.1147/sj.41.0025

OpenEXR Development Team. (2023). *OpenEXR Technical Introduction*. https://openexr.com/en/latest/TechnicalIntroduction.html