Final Grafica

Contents

1 Jeremy

1.1 Pregunta 4: Marching Cubes

Question: Describe the marching cubes algorithm and how it can be used to generate isosurfaces in a 3D scalar field. What are the main challenges and how are they resolved? Notice that the adaptative sampling that we used in class is not what is being asked in this question.

Answer:

El algoritmo Marching Cubes es un método ampliamente utilizado para extraer una malla poligonal de una isosuperficie a partir de un campo escalar tridimensional. Fue desarrollado por Lorensen y Cline en 1987 y desde entonces se ha convertido en una técnica estándar en gráficos por computadora y visualización.

1.1.1 Resumen

- **Entrada:** Un campo escalar 3D definido en una cuadrícula de puntos.
- **Salida:** Un conjunto de polígonos (generalmente triángulos) que aproximan la isosuperficie donde el campo escalar iguala a un valor especificado (valor iso).

1.1.2 Algoritmo

1. **Procesamiento de la Cuadrícula y los Cubos:**
   - El algoritmo procesa el campo escalar en una base de cubos, donde cada cubo está definido por ocho vértices (esquinas del cubo) en la cuadrícula.
   - Cada vértice del cubo puede estar dentro o fuera de la isosuperficie, dependiendo de si su valor escalar está por debajo o por encima del valor iso.
2. **Clasificación de los Vértices:**
   - Clasificar cada vértice del cubo como dentro (1) o fuera (0) de la isosuperficie basándose en el valor iso.
3. **Intersecciones de los Bordes:**
   - Determinar dónde la isosuperficie intersecta los bordes del cubo. Esto se hace mediante interpolación lineal entre los valores escalares en los vértices del cubo.
4. **Índice de Caso:**
   - Utilizar la clasificación de los vértices para generar un índice de caso (un número binario de 8 bits que representa qué vértices están dentro de la isosuperficie).
   - Este índice se usa para buscar en una tabla precomputada que define cómo conectar los bordes intersectados para formar triángulos.
5. **Generación de Triángulos:**
   - A partir de las intersecciones de los bordes y la tabla de búsqueda, generar los triángulos que forman la isosuperficie dentro del cubo.
6. **Repetir para Todos los Cubos:**
   - Aplicar los pasos anteriores a cada cubo en la cuadrícula para construir toda la isosuperficie.

1.1.3 Desafíos y Soluciones

- **Ambigüedades:**
  - El algoritmo Marching Cubes original puede enfrentar ambigüedades en ciertas configuraciones donde existen múltiples triangulaciones válidas.
  - **Solución:** Extensiones como Asymptotic Decider pueden resolver ambigüedades considerando los puntos medios de los bordes del cubo.
- **Consistencia:**
  - Asegurar que la malla generada sea hermética (es decir, sin huecos ni grietas) es crítico.
  - **Solución:** Diseño cuidadoso de las tablas de búsqueda para asegurar que los cubos adyacentes produzcan triángulos coincidentes.
- **Rendimiento:**
  - El algoritmo puede ser intensivo en cómputo, especialmente para cuadrículas grandes.
  - **Solución:** Optimizaciones como procesar solo los cubos que intersectan la isosuperficie y aprovechar el procesamiento paralelo pueden mejorar el rendimiento.

1.2 Pregunta 11: Mesh Parametrization

Question: What is mesh parametrization? Describe (for dummies) 4 common methods for parametrizing a mesh.

Answer:

La parametrización de mallas es un proceso que mapea una superficie 3D a un plano 2D. Esto es útil en diversas aplicaciones como el mapeo de texturas, la remallado y el morphing. Al adjuntar un sistema de coordenadas a la superficie, los problemas complejos de modelado 3D se pueden simplificar en un espacio 2D.

4 Métodos Comunes para Parametrizar una Malla:

1. **Mapeo Baricéntrico**:
   - **Descripción:** Este método se basa en el teorema de mapeo baricéntrico de Tutte. Involucra fijar los vértices de la frontera de la malla en un polígono convexo y luego encontrar las posiciones de los vértices internos de manera que sean una combinación convexa de sus vecinos.
   - **Para Principiantes:** Imagina estirar una malla como si fuera una hoja de goma para que su frontera quede plana sobre una mesa. Los puntos internos se colocan donde se equilibran con sus vecinos.
   - **Ejemplo:** Usado en la creación de mapas de texturas donde se fija la frontera y se ajustan los puntos internos para mantener la forma.
2. **Mapeo Conforme**:
   - **Descripción:** Los mapas conformes preservan los ángulos, haciendo que la parametrización conserve la forma localmente. Este método utiliza técnicas de análisis complejo para asegurar que las pequeñas formas en la malla se mapeen a formas similares en el plano.
   - **Para Principiantes:** Imagina colocar una tela flexible pero no elástica sobre la malla, donde la tela se ajusta pero no distorsiona las formas locales.
   - **Ejemplo:** Usado en aplicaciones que requieren una preservación precisa de los ángulos como en la imaginería médica.
3. **Mapas Conformes de Mínimos Cuadrados (LSCM)**:
   - **Descripción:** Este método minimiza la desviación de la conformidad resolviendo un problema de mínimos cuadrados. Asegura que la parametrización sea lo más conforme posible ajustando los vértices de la malla para minimizar la distorsión de los ángulos.
   - **Para Principiantes:** Piensa en tratar de aplanar un mapa arrugado presionándolo uniformemente para reducir los pliegues, asegurándote de que las formas locales estén lo menos distorsionadas posible.
   - **Ejemplo:** Comúnmente usado en herramientas de modelado 3D como Blender para el mapeo de texturas.
4. **Aplanamiento Basado en Ángulos (ABF)**:
   - **Descripción:** El ABF ajusta los ángulos de los triángulos en la malla para minimizar la distorsión. Asegura que la suma de los ángulos alrededor de cada vértice en el espacio de parámetros coincida con la suma de los ángulos en el espacio 3D.
   - **Para Principiantes:** Imagina ajustar las esquinas de cada triángulo en un modelo de papel para que cuando se aplane, los ángulos coincidan perfectamente con la forma original 3D.
   - **Ejemplo:** Usado en la creación de modelos 3D para animaciones donde mantener la forma original es crucial.

1.3 Pregunta 15: Ray Tracing

Question: Explain the ray tracing method for image generation. Mention advantages of this method when compared to the Painter's algorithm.

Answer:

Ray tracing is a rendering technique used to generate images by simulating the way light interacts with objects in a virtual environment. The process involves tracing the path of rays from the eye (or camera) through the pixels of the image plane and into the scene. When a ray intersects an object, calculations are performed to determine the color of the pixel based on the material properties of the object, the light sources, and the contribution of other objects in the scene through reflections, refractions, and shadows.

Advantages of Ray Tracing compared to the Painter's Algorithm:

1. **Realistic Lighting and Shadows**:
   - **Ray Tracing:** Accurately simulates global illumination, including reflections, refractions, and shadows, leading to highly realistic images. It accounts for light interactions such as caustics and diffuse interreflections.

- **Painter's Algorithm:** Primarily a visibility algorithm that sorts and renders polygons from back to front, not accounting for complex light interactions. Shadows and reflections need to be manually added, often resulting in less realism.

2. **Handling Complex Geometries**:
   - **Ray Tracing:** Can handle complex scenes with intricate geometries and overlapping objects efficiently due to its mathematical approach to tracing rays.
   - **Painter's Algorithm:** Struggles with complex scenes where objects overlap or intersect, as it relies on sorting polygons which can be computationally intensive and error-prone.
3. **Accuracy in Visual Effects**:
   - **Ray Tracing:** Naturally accommodates realistic visual effects such as transparency, translucency, and optical effects like chromatic aberration.
   - **Painter's Algorithm:** Requires additional algorithms and processing to approximate these effects, often leading to less accurate results.
4. **Dynamic Scenes**:
   - **Ray Tracing:** Better suited for dynamic scenes where objects move or lighting changes, as it recalculates interactions on a per-ray basis.
   - **Painter's Algorithm:** Less flexible in dynamic scenes, as the sorting order of polygons needs to be recalculated, which can be computationally expensive.

1.4 Pregunta 16: Radiosity

Question: Describe the process of rendering using the radiosity algorithm. Explain why we say that the radiosity method is viewpoint-independent.

Answer:

Rendering with the Radiosity Algorithm

The radiosity method for rendering is a technique that focuses on simulating the physics of light transport within a scene, particularly for environments with Lambertian surfaces. Unlike other rendering methods that trace rays from the viewpoint to determine visible surfaces, radiosity begins with the light emitted from sources and computes its interactions with surfaces in the scene.

1. **Surface Partitioning**:
   - The scene's surfaces are divided into small, typically rectangular patches.
   - Each patch is assumed to have constant illumination and can be represented by a single radiosity value.
2. **Lambertian Assumption**:
   - Surfaces are considered Lambertian reflectors, meaning they reflect light uniformly in all directions.
   - This simplifies the light transport calculations since the reflected radiance is independent of the viewing direction.
3. **Form Factor Calculation**:
   - Form factors determine how much light a patch receives from other patches.
   - They are computed based on the geometry of the scene, taking into account the visibility and orientation of the patches relative to each other.
4. **Radiosity Equation**:
   - The radiosity of a patch B_j is given by the equation:

$$B_j = E_j + \rho_j \sum_k F_{jk} B_k$$

   where E_j is the emitted radiance, ρ_j is the reflectivity, and F_{jk} is the form factor from patch k to patch j.
5. **Solving the Radiosity Equation**:
   - The equation is solved iteratively or using linear algebra techniques to find the radiosity values for all patches.
   - Once the radiosities are computed, they represent the steady-state distribution of light in the scene.

6. **Rendering the Image**:
    - The computed radiosity values are used to render the final image.
    - Since the radiosity values represent the surface radiance, the rendering process involves displaying these values directly or interpolating between them for smoother results.

Viewpoint Independence
The radiosity method is considered viewpoint-independent because it computes the distribution of light within a scene without regard to the viewer's position. This characteristic arises from the following factors:

- **Global Illumination**:
    - Radiosity calculates the light interactions and reflections within the entire scene, resulting in a global solution for surface radiance.
- **Patch-Based Computation**:
    - The method focuses on the interaction between patches and the overall energy balance, rather than tracing rays from a specific viewpoint.
- **Precomputed Lighting**:
    - Once the radiosity values are determined, they can be reused for rendering the scene from any viewpoint, as the light distribution is already known.
    - This makes the method particularly efficient for static scenes where the lighting does not change, as the computed radiosities do not need to be recalculated for different viewpoints.

1.5 Pregunta 18: Practical Question (Ray tracing)

Develop an algorithm for calculating the intersection between a ray and a triangle. Context: the algorithm will be applied in a raytracer.

Algorithm for Ray-Triangle Intersection: To calculate the intersection between a ray and a triangle, you can use the Möller-Trumbore intersection algorithm. This algorithm is widely used and provides an efficient solution for this problem.

You can find the detailed explanation and implementation of the Möller-Trumbore algorithm in the following link: Möller-Trumbore Intersection Algorithm.

```
std::optional<vec3> ray_intersects_triangle(const vec3 &ray_origin,
                                            const vec3 &ray_vector,
                                            const triangle3& triangle)
{
    constexpr float epsilon = std::numeric_limits<float>::epsilon();

    vec3 edge1 = triangle.b - triangle.a;
    vec3 edge2 = triangle.c - triangle.a;
    vec3 ray_cross_e2 = cross(ray_vector, edge2);
    float det = dot(edge1, ray_cross_e2);

    if (det > -epsilon && det < epsilon)
        return {}; // This ray is parallel to this triangle.

    float inv_det = 1.0 / det;
    vec3 s = ray_origin - triangle.a;
    float u = inv_det * dot(s, ray_cross_e2);

    if (u < 0 || u > 1)
        return {};

    vec3 s_cross_e1 = cross(s, edge1);
    float v = inv_det * dot(ray_vector, s_cross_e1);

    if (v < 0 || u + v > 1)
        return {};

    // At this stage we can compute t to find out where the intersection point is on the line.
    float t = inv_det * dot(edge2, s_cross_e1);

    if (t > epsilon) // ray intersection
        return vec3(ray_origin + ray_vector * t);
    else // This means that there is a line intersection but not a ray intersection.
        return {};
```

```
5 }
```

Listing 1: Intersección de rayos con triángulos

1.6 Pregunta 21: Computer Vision (Stereo Cameras)

Question

You have a stereo camera system with a baseline (distance between the two cameras) of 10 cm. The focal length of each camera is 50 mm. The pixel coordinates of a point in the left image are $(x_L, y_L) = (150, 200)$, and in the right image, they are $(x_R, y_R) = (130, 200)$. The size of each pixel is 0.01 mm. Calculate the 3D coordinates (X, Y, Z) of the point. Provide your calculations and justifications.

Answer

Cálculo de Coordenadas 3D con un Sistema de Cámara Estéreo

Datos Dados:

- Línea base (b) = 10 cm = 100 mm
- Longitud focal (f) = 50 mm
- Coordenadas del píxel en la imagen izquierda $(x_L, y_L) = (150, 200)$
- Coordenadas del píxel en la imagen derecha $(x_R, y_R) = (130, 200)$
- Tamaño de cada píxel = 0.01 mm

Fórmulas Utilizadas:

$$\text{Disparidad} \quad d = x_L - x_R \tag{1}$$

$$\text{Profundidad} \quad Z = \frac{bf}{d} \tag{2}$$

$$\text{Coordenada Horizontal} \quad X = \frac{b(x_L + x_R)}{2d} \tag{3}$$

$$\text{Coordenada Vertical} \quad Y = \frac{by}{d} \tag{4}$$

Cálculos:

1. **Calcular la disparidad**:

$$d = x_L - x_R \tag{5}$$
$$= 150 - 130 \tag{6}$$
$$= 20 \text{ píxeles} \tag{7}$$

2. **Convertir la disparidad a milímetros**:

$$d_{\text{mm}} = d \times \text{tamaño del píxel} \tag{8}$$
$$= 20 \times 0.01 \tag{9}$$
$$= 0.2 \text{ mm} \tag{10}$$

3. **Calcular** Z:

$$Z = \frac{bf}{d_{\text{mm}}} \tag{11}$$
$$= \frac{100 \text{ mm} \times 50 \text{ mm}}{0.2 \text{ mm}} \tag{12}$$
$$= \frac{5000 \text{ mm}^2}{0.2 \text{ mm}} \tag{13}$$
$$= 25000 \text{ mm} \tag{14}$$
$$= 2500 \text{ cm} \tag{15}$$

4. **Calcular** X:

$$X = \frac{b(x_L + x_R)}{2d_{\text{mm}}} \tag{16}$$
$$= \frac{100 \text{ mm} \times (150 + 130)}{2 \times 0.2 \text{ mm}} \tag{17}$$
$$= \frac{100 \text{ mm} \times 280}{0.4 \text{ mm}} \tag{18}$$
$$= \frac{28000 \text{ mm}^2}{0.4 \text{ mm}} \tag{19}$$
$$= 70000 \text{ mm} \tag{20}$$
$$= 7000 \text{ cm} \tag{21}$$

5. **Calcular** Y:
Dado que $y_L = y_R = 200$ píxeles. Convirtiendo esto a milímetros:

$$y_{\text{mm}} = 200 \times 0.01 \tag{22}$$
$$= 2 \text{ mm} \tag{23}$$

Entonces,

$$Y = \frac{by_{\text{mm}}}{d_{\text{mm}}} \tag{24}$$
$$= \frac{100 \text{ mm} \times 2 \text{ mm}}{0.2 \text{ mm}} \tag{25}$$
$$= \frac{200 \text{ mm}^2}{0.2 \text{ mm}} \tag{26}$$
$$= 1000 \text{ mm} \tag{27}$$
$$= 100 \text{ cm} \tag{28}$$

Conclusión:
Las coordenadas 3D (X, Y, Z) del punto son:

$$(X, Y, Z) = (7000 \text{ cm}, 100 \text{ cm}, 2500 \text{ cm}) \tag{29}$$

1.7 Pregunta 22: Computer Vision (Extracting and Matching Features)

Explain how the Scale-Invariant Feature Transform (SIFT) algorithm extracts keypoints from an image and de- scribe how these keypoints are matched across different images. Explain also how the Harris Corners are extracted from different images and how can be matched.

1.7.1 Scale-Invariant Feature Transform (SIFT)

Keypoint Extraction

1. **Scale-Space Extrema Detection:**
   - The SIFT algorithm constructs a scale space by generating Gaussian blurred images at different scales.
   - The Difference of Gaussians (DoG) is calculated by subtracting adjacent Gaussian blurred images.
   - Keypoints are identified as local extrema in the DoG images, i.e., points that are either maximum or minimum compared to their neighbors in both scale and space.
2. **Keypoint Localization:**
   - The algorithm refines the location of each keypoint by fitting a quadratic function to the local sample points for accurate position, scale, and contrast.
   - Low-contrast keypoints and those poorly localized along edges are discarded.
3. **Orientation Assignment:**
   - For each keypoint, an orientation is assigned based on the local image gradient directions.
   - This orientation provides invariance to image rotation.

4. **Keypoint Descriptor:**
   - A descriptor is created for each keypoint based on the local image gradients.
   - The descriptor is typically a 128-dimensional vector (formed from 16 sub-regions, each with an 8-bin orientation histogram).
   - The descriptor is normalized to achieve invariance to changes in illumination.

Keypoint Matching

- **Descriptor Matching:**
  - Keypoints between images are matched by comparing their descriptors.
  - The Euclidean distance between descriptors is used as a similarity measure.
- **Lowe's Ratio Test:**
  - A common approach is to use Lowe's ratio test, which compares the distance of the closest neighbor to the distance of the second closest neighbor.
  - Matches are accepted if the ratio is below a certain threshold (e.g., 0.8), reducing the likelihood of false matches.

1.7.2 Harris Corner Detection

Corner Extraction

1. **Gradient Calculation:**
   - Compute the image gradients (I_x and I_y) for each pixel using a Sobel operator or similar method.
2. **Structure Tensor:**
   - Calculate the structure tensor M for each pixel:

$$M = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$$

3. **Corner Response Function:**
   - Compute the Harris response (R) using the determinant and trace of the structure tensor:

$$R = \det(M) - k(\text{trace}(M))^2$$

   - k is a sensitivity parameter, typically between 0.04 and 0.06.
4. **Non-Maximum Suppression:**
   - Identify local maxima in the Harris response image.
   - Thresholding is applied to select significant corners.

Corner Matching

- **Descriptor Calculation:**
  - Calculate descriptors (e.g., using SIFT, SURF, or simple intensity patches) around detected corners to provide a feature vector.
- **Descriptor Matching:**
  - Match corners across images by comparing these descriptors using a distance metric (e.g., Euclidean distance).
  - Similar to SIFT, techniques like Lowe's ratio test can be used to validate matches.

1.7.3 Comparison and Use Cases

- **SIFT:** Effective for matching objects in images taken under different conditions (scale, rotation, illumination changes). Suitable for object recognition, image stitching, and 3D reconstruction.
- **Harris Corners:** Efficient for detecting corners, which are often used in tracking and motion analysis. Less robust to scale and rotation changes compared to SIFT.

Both methods involve identifying distinctive points in images and creating descriptors to match these points across different images. The main difference lies in their robustness and invariance properties, with SIFT providing a more comprehensive approach at the cost of higher computational complexity.

2 Renato

2.1 Pregunta 1: Surface Representation

Question: Explain the differences between explicit, implicit, and parametric surface representations. Provide examples of each and discuss their advantages and disadvantages.

Answer:

Explicit Representation Define a surface as a function of 2 independent variables, which returns a single dependent variable. Example: $f(x, y) = mx + by + c = z$.

- **Advantages:** Useful for computing intersections, intuitive vision.
- **Disadvantages:** Cannot easily represent surfaces that cannot be written as a function (x, y), such as a torus or a sphere.

Implicit Representation Define a surface as a set of points (x, y, z) that satisfy an equation of this form $f(x, y, z) = 0$.

- **Advantages:** Easily represents surfaces that cannot be written as a function (x, y). Useful for a torus.
- **Disadvantages:** Not as useful for representing intersections with rays or other geometric forms.

Parametric Representation Define a surface as a vector function of two parameters (u, v). Normally of this form $r(u, v) = (x(u, v), y(u, v), z(u, v))$. Where x, y, and z are functions of parameters u and v.

- **Advantages:**
  - Easily represents surfaces that cannot be written as a function (x, y). Useful for a torus.
  - Useful for texture mapping, as the surface can be parameterized.
  - Useful for polygonal approximations.
- **Disadvantages:**
  - Parameterizing the equations is less intuitive than implicit and explicit equations.
  - Computing intersections for ray tracing or geometric forms is more complex than with explicit representations.

2.2 Pregunta 7: Mesh Data Structures 2

Question: Consider meshes represented with the data structures described in the previous question. Describe algorithms for many useful operations. Analyze conceptual simplicity and computational complexity of each of them. The following list serves as inspiration for what the useful operations can be. By no means this list should be considered as an extensive list. Feel free to think about other potential operations.

Answer:

- Iterate through the neighbors of a vertex: Traverse all vertices connected to a given vertex.
- Split an edge: Insert a new vertex in the middle of an edge, subdividing adjacent triangles.
- Merge coincident vertices: Combine vertices at the same spatial position to eliminate duplicates and clean up the mesh.
- Remove a vertex and retriangulate adjacent faces: Remove a vertex and adjust surrounding faces to maintain mesh topology.
- Compute vertex normal: Calculate the direction perpendicular to the surface at a vertex by averaging adjacent face normals.
- Identify and fix non-manifold edges: Detect edges shared by more than two faces and correct them to maintain a valid manifold mesh.
- Refine a face: Subdivide a face into smaller faces.
- Simplify a face: Merge faces or vertices to reduce mesh complexity while preserving overall shape.
- Calculate area of a face: Compute the surface area of a polygonal face for geometrical analysis.
- Measure edge length: Determine the length of an edge.
- Identify isolated vertices: Find vertices not connected to any faces (which may indicate errors or incomplete data).
- Swap face vertices: Change the order of vertices in a face.
- Adjust vertex positions.
- Calculate curvature at a vertex: Determine the rate of change of surface normals at a vertex to assess surface features.
- Traverse face adjacency: Iterate over faces sharing an edge.

2.3 Pregunta 8: Subdivision Surfaces

Question:

Explain the concept of subdivision surfaces and how they are used to generate smooth surfaces from polygonal meshes. Provide an example of a practical application.

Answer:

El proceso de subdivisión comienza con una malla poligonal base, conocida como la mesh control. Esta malla inicial define la forma y topología general de la superficie. Luego, se aplica un algoritmo de subdivisión a esta malla, que realiza los siguientes pasos:

1. Cada cara poligonal se subdivide en caras más pequeñas, generalmente agregando nuevos vértices en los puntos medios de los bordes y las caras.
2. Las posiciones de los nuevos vértices se calculan en función de las posiciones de los vértices circundantes en la malla original, utilizando un conjunto de reglas matemáticas definidas por el esquema de subdivisión específico.
3. Los vértices originales también pueden moverse ligeramente para mejorar la suavidad de la superficie resultante.

Este proceso de subdivisión se puede repetir de forma iterativa, donde cada paso crea una malla más densa y suave que mejor aproxima la "superficie límite" final. Algunos esquemas de subdivisión comunes son Catmull-Clark, Loop.

2.4 Aplicación Práctica

Las subdivision surfaces se utilizan para realizar animaciones para modelar formas orgánicas y complejas.

- Suavidad
- Flexibilidad

2.5 Pregunta 9: Catmull-Clark Algorithm

Question:

Describe the Catmull-Clark algorithm for surface subdivision. Explain how new vertices are generated and how faces and edges are updated.

Answer:

1. **Nuevos puntos en las caras**
   (a) Para cada **cada** de la malla original se crea un punto nuevo. Este punto se define como el promedio de todos los vértices originales que conforman esa cara.
2. **Nuevos puntos en las aristas**
   (a) Para cada arista de la malla original se crea un nuevo en la arista nueva. Este punto también es el promedio de los extremos de la arista.
3. **Nuevos puntos en los vértices**
   (a) Para cada vértice (P) de la malla original se calcula uno nuevo. Toma el promedio (F) de todos los puntos de todos los nuevos puntos generados por las caras que tocan P, y el promedio (R) de todos los nuevos puntos en las aristas que tocan P. En resumen se calcula de esta forma $\frac{(n-3)P+F+2R}{n}$. Donde n es el número de caras que tocan el vértice original P.
4. **Después de calcular todos los nuevos puntos. El mesh original se actualiza:**
   (a) Se crean nuevas aristas conectando cada nuevo punto generado por una cara a los nuevos puntos generados por aristas que definen esas caras.
   (b) Se crean nuevas aristas conectando cada nuevo punto generado por vértices a los nuevos puntos generados por aristas que inciden en ese vértice.
   (c) Finalmente, se definen las nuevas caras como cuadriláteros por estas nuevas aristas.

2.6 Pregunta 10: Texture Mapping via Barycentric Coordinates

Question: Explain the process of texture mapping on a triangle using barycentric coordinates.

Answer:

Primero, definiremos las coordenadas baricéntricas para representar un punto dentro de un triángulo. Cualquier punto q dentro de un triángulo p_1, p_2, p_3 puede expresarse como:

$$q = \alpha p_1 + \beta p_2 + \gamma p_3$$

donde $\alpha + \beta + \gamma = 1$.

Luego, definimos la interpolación de las coordenas de la textura.

Para cada coordenada de la textura (u, v) la asignamos a cualquier punto q dentro del triángulo con las coordenadas baricéntricas α, β, γ.

$$u = \alpha u_1 + \beta u_2 + \gamma u_3$$

$$v = \alpha v_1 + \beta v_2 + \gamma v_3$$

donde $(u_1, v_1), (u_2, v_2), (u_3, v_3)$ son las coordenadas de la textura en los vértices del triángulo.

Las textura en un espacio 3D puede causar una distorsión. Esto se puede corregir, usando el canal alpha en una codificación de colores RBGA.

2.7 Pregunta 12: Topological Operators and Simplification

Question: Explain what topological operators are in the context of meshes and how they can be used in mesh simplification. Provide two examples of topological operators.

Answer:

Los operadores topológicos son operaciones fundamentales que se pueden realizar en modelos de meshes para simplificar o modificar su estructura preservando propiedades topológicas generales. Son importantes en algoritmos de simplificación de meshes que tienen como objetivo reducir la complejidad de un modelo manteniendo propiedades geométricas y topológicas esenciales.

Ejemplos

- Eliminación de vértices: Elimina un arista y sus vértices relacionadas. Y funciona las caras vecinas en una sola cara.
- Division de aristas: Este es el operador inverso al colapso de aristas, introduce un nuevo vértice en un arista, dividiendo la arista y sus caras incidentes en dos. Puede utilizarse para aumentar la resolución de una malla o para aumentar desahacer una operación de colapso de arista.

2.8 Pregunta 13: Implicit Surfaces

Question: Define what an implicit surface is and describe how it can be used in the modeling of more complex shapes. What are the advantages and challenges of using implicit surfaces?

Answer:

Una superficie implícita es una figura geométrica definida por una función que devuelve un número escalar $f : \mathbb{R}^3 \to \mathbb{R}$ que asigna un valor a cada punto en un espacio tridimensional.

Esta función determina si el punto está dentro, fuera o en la superficie. La superficie se define como:

$$f(x, y, z) = 0$$

Son particularmente útiles para modelar superficies complejas porque pueden representar aristas a cualquier resolución debido a que no están limitados a alguna forma. Esto es posible con técnicas como Marching Cubes.

Ventajas

- Estas superficies fácilmente pueden ser trazadas usando *ray-tracing*.
- Son flexibles a ser usadas a una gran variedad de formas, es ideal para realizar visualizaciones y animaciones.

Retos

- La visualización requiere de alguna técnica adicional para que se logre, ya que esta representación no nos da directamente la visualización de la superficie. Podemos usar *marching cubes*.
- Mientras más precisa queramos que sea la superficie la complejidad computacional será mayor.

2.9 Pregunta 20: Pinhole Camera Model

Question:
Design a pinhole camera system to capture a clear image of an object located 3 meters away. The camera has a 35 mm wide sensor and we want the object to occupy 70% of the sensor's width. What should be the focal length of the pinhole camera?

Answer: Para determinar la distancia focal de la cámara, usamos:

$$\frac{f}{d} = \frac{w}{W}$$

Donde:

- f: es la distancia focal de la cámara
- d: es la distancia al objeto = 3 metros o 3000 mm
- w: ancho del sensor = 35 mm
- W: es el ancho del objeto en la escena. $W = 0.7 \times 35 = 24.5\,\text{mm}$. ya que deseamos que el objeto ocupe el 70% del ancho del sensor.

$$f = \frac{d \times w}{W} = \frac{3000 \times 35}{24.5} = 4285.71\,\text{mm}$$

Por lo tanto la distancia focal del objeto es aproximadamente 4 metros.

3 Luis

3.1 Pregunta 2: Marching Squares

Question: Describe the marching squares algorithm and its application in generating contours of a scalar function defined on a square grid. Include a graphical example illustrating the process. Notice that the adaptive sampling that we used in class is not what is being asked in this question.

Answer:
Marching Squares es un algoritmo enfocado en la extracción de líneas de contorno de un campo escalar bidimensional (x, y). El algoritmo consta de 6 pasos generales explicados a continuación:

1. **Inicialización:**
   - Definir un campo escalar en una cuadrícula bidimensional. Cada celda de la cuadrícula tiene cuatro puntos de esquina con valores escalares asociados.
   - Determinar un valor umbral que definirá las líneas de contorno.
2. **Asignación de Estados a los Vértices:**
   - Para cada celda en la cuadrícula, comparar los valores escalares de los cuatro puntos de esquina con el valor umbral.
   - Asignar un estado binario (0 si el valor escalar está por debajo del umbral, 1 si está por encima) a cada punto de esquina de la celda.
3. **Generar el Patrón Binario de la Celda:**
   - Formar un patrón binario de 4 bits para cada celda combinando los estados de los cuatro vértices. Cada patrón representa una posible configuración de la celda respecto al umbral.
   - Hay 16 posibles patrones (2^4 combinaciones).
4. **Buscar en la Tabla de Casos:**
   - Usar el patrón binario para buscar en una tabla de casos predefinidos que describe cómo los segmentos de contorno atraviesan la celda.
   - Cada patrón binario corresponde a uno de los 16 casos, que indican las posiciones relativas de los segmentos de contorno dentro de la celda.
5. **Interpolación de Posiciones:**

- Para cada borde de la celda donde el contorno intersecta, calcular la posición exacta del punto de intersección mediante interpolación lineal entre los valores escalares de los vértices de la celda.
- La interpolación se hace para encontrar las coordenadas precisas en las que el contorno cruza el borde de la celda.

6. **Dibujo de los Segmentos de Contorno:**
   - Dibujar los segmentos de contorno dentro de la celda según el caso determinado por el patrón binario y las posiciones interpoladas.
   - Conectar estos segmentos a través de celdas adyacentes para formar líneas de contorno continuas a lo largo de la cuadrícula.

3.2 Pregunta 3: Marching Triangles

Question: Compare and contrast the marching squares and marching triangles algorithms. Discuss situations in which each would be more appropriate to use.

Answer:

Tanto el Marching Squares como el Marching Triangles son algoritmos utilizados para la extracción de isolíneas de campos escalares. Sin embargo, sus principales diferencias recaen en los siguientes aspectos:

Aspecto	Marching Squares	Marching Triangles
Grid Type	Cuadrícula Cuadrada	Cuadrícula Triangular
Cell Shape	4 puntos de vértices	3 puntos de vértices
Binary Pattern	16 posibles patrones	8 patrones posibles
Interpolación	En los bordes de los cuadrados	En los bordes de los triángulos
Conectividad de Segmentos	Entre las celdas adyacentes	Entre los triángulos adyacentes
Ventajas	Mejor manejo por cuadrículas regulares Ideal para procesamiento de imágenes Simple e intuitivo	Manejo de geometrías complejas Ideal para procesamiento de datos no estructurados Menos ambigüedad en intersecciones de contornos
Desventajas	Limitado a grillas cuadradas Puede producir resultados ambiguos	Más complejo que marching squares Menos eficiente en grillas regulares
Aplicaciones	Procesamiento de imágenes Mapas topográficos	Análisis de elementos finitos Modelo geológico y ambiental

El algoritmo de Marching Squares es más adecuado para el procesamiento de imágenes y mapas topográficos debido a su simplicidad y efectividad en cuadrículas regulares. Por otro lado, el algoritmo de Marching Triangles es más apropiado para el análisis de elementos finitos y modelos geológicos o ambientales donde las geometrías complejas y los datos no estructurados son comunes.

3.3 Pregunta 5: OFF and PLY Formats

Question: Explain the structure of the OFF and PLY formats for mesh representation. What are the advantages and disadvantages of each of them, compared to the other?

Answer:

Estructura de OFF:

- **Header:** La primera linea lleva el nombre del formato **OFF**, seguido del número de dimensiones del archivo. Finalmente se indica el número de vértices, número de caras y número de aristas.
- **Vertices:** Se ingresan las posiciones x, y, z de los vértices y opciones adicionales como colores, coordenadas de textura si los prefijos están en la cabecera.
- **Faces:** Cada linea indica el número de vértices a utilizar para la cara, seguido de los vértices que componen dicha cara, se puede agregar el color de la cara luego de los vértices.

Estructura de PLY:

- **Header:** La primera linea declara el formato del archivo **PLY**, la siguiente linea define si el archivo está en formato ascii o binario, le siguen comentarios y el número de vértices, el tipo de variables utilizadas y el número de caras, en ese orden. Se puede agregar los colores de los vértices y de las caras.
- **Vertices:** Lista de posiciones x, y, z de cada vértices.
- **Faces:** Lista de números de vértices utilizados para formar cada cara.

Feature	OFF	PLY
Simplicity	Very simple, easy to parse	More complex due to support for multiple properties
Human-readable	Yes, text-based	Yes for ASCII, no for binary
Support for attributes	No support for attributes	Supports attributes like colors, normals, texture coordinates
Efficiency	Less efficient for large meshes	More efficient with binary format
Ease of editing	Easy to edit manually	Easy for ASCII, difficult for binary
Usage scenarios	Small to medium-sized, simple meshes	Large, complex meshes with additional attributes

3.4 Pregunta 6: Mesh Data Structures 1

Question: Describe at least three different data structures that can be used to represent meshes. Include in your description: - half-edge - winged-edge - some easy and quick representation that you might have used and is not one of the two previous.
Answer:

3.4.1 Half-Edge Data Structure

Description: The half-edge data structure is designed to efficiently represent the connectivity of a polygon mesh, especially for surfaces. It explicitly stores the relationships between edges, vertices, and faces, enabling fast traversal and manipulation.
Structure:

- **Half-Edge:** Each edge is split into two half-edges, each with a direction. A half-edge points to its next and previous half-edges in the face, its opposite half-edge, its starting vertex, and the face it borders.
- **Vertex:** Stores coordinates and a reference to one of its outgoing half-edges.
- **Face:** Stores a reference to one of its half-edges.

Advantages:

- **Efficient Traversal:** Enables efficient traversal of the mesh for algorithms that need to navigate through vertices, edges, and faces.
- **Easy Manipulation:** Simplifies operations like edge flipping, splitting, and collapsing.

Disadvantages:

- **Complexity:** More complex to implement and understand compared to simpler structures like polygon soup.
- **Memory Overhead:** Requires more memory to store the connectivity information.

Use Cases:

- Ideal for applications needing dynamic mesh editing, such as CAD systems and computer graphics.

3.4.2 Winged-Edge Data Structure

Description: The winged-edge data structure represents the connectivity of a mesh by storing information about the edges and their adjacent elements. It focuses on the edges and their relationships with the adjacent vertices and faces.
Structure:

- **Edge:** Stores references to its two vertices, its two adjacent faces, and the four neighboring edges (called "wings") that share the same vertices.
- **Vertex:** Stores coordinates and a reference to one of its incident edges.
- **Face:** Stores a reference to one of its edges.

Advantages:

- **Efficient Edge Operations:** Facilitates efficient edge-centric operations, such as finding neighboring edges and faces.
- **Detailed Connectivity:** Provides detailed connectivity information, making it useful for complex mesh manipulations.

Disadvantages:

- **Complexity:** More complex and harder to implement than half-edge and polygon soup structures.
- **Memory Usage:** Requires significant memory to store all the references.

Use Cases:

- Suitable for applications requiring extensive edge manipulation and complex mesh operations, such as geometric modeling and mesh optimization.

3.4.3 Polygon Soup

Description: The polygon soup data structure represents a mesh as an unordered collection of polygons (typically triangles or quadrilaterals) without any explicit connectivity information. Each polygon is defined independently by its vertices.

Structure:

- **Polygon:** Each polygon is defined by a list of vertex indices.
- **Vertex:** Stores coordinates.

Advantages:

- **Simplicity:** Very simple to implement and understand.
- **Memory Efficiency:** Minimal memory overhead since it does not store connectivity information.

Disadvantages:

- **Lack of Connectivity:** No explicit connectivity information, making traversal and certain operations (like edge-based algorithms) inefficient.
- **Redundancy:** Potentially redundant storage of vertices if shared between multiple polygons.

Use Cases:

- Suitable for simple applications where mesh connectivity is not critical, such as rendering static models and basic mesh storage.

3.4.4 Comparison

Data Structure	Connectivity Information	Memory Usage	Complexity	Efficiency in Traversal	Use Cases
Half-Edge	Detailed	High	High	Efficient	Dynamic mesh editing, CAD s
Winged-Edge	Very Detailed	Very High	Very High	Efficient	Geometric modeling, mesh opt
Polygon Soup	None	Low	Low	Inefficient	Simple applications, rendering

3.5 Pregunta 14: Interval Arithmetic

Question: Explain the concept of interval arithmetic and its potential application in marching squares and marching cubes.

Answer:

Concept: Interval arithmetic is a mathematical approach where numbers are represented as intervals rather than precise values. Each interval includes a lower and an upper bound, indicating the range within which the actual value lies. This method is particularly useful for handling uncertainties and ensuring that operations account for all possible values within the specified ranges.

Basic Operations:

1. **Addition:**
$$[a, b] + [c, d] = [a + c, b + d]$$
2. **Subtraction:**
$$[a, b] - [c, d] = [a - d, b - c]$$
3. **Multiplication:**
$$[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$$
4. **Division:**
$$[a, b] \div [c, d] = [a, b] \times \left[\frac{1}{d}, \frac{1}{c}\right] \quad (\text{if } 0 \notin [c, d])$$

Potential Application in Marching Squares and Marching Cubes

- **Marching Squares**:
  - **Handling Uncertainties:** Interval arithmetic can manage uncertainties in the scalar field values. When the values at grid points are not exact (due to measurement errors or other factors), representing these values as intervals ensures that the algorithm accounts for all possible scenarios within the intervals.
  - **Intersection Tests:** During the contour extraction, each vertex's scalar value is an interval. The intersection tests (whether a contour crosses an edge) can be performed with interval comparisons, ensuring robustness against uncertainties.
  - **Improved Accuracy:** By incorporating interval arithmetic, the contours generated by Marching Squares can be more accurate and reliable, especially in scientific and engineering applications where precision is crucial.
- **Marching Cubes**:
  - **3D Contour Extraction:** Similar to Marching Squares, interval arithmetic can be applied to the Marching Cubes algorithm for 3D contour extraction. The scalar field values at the vertices of the cubes can be represented as intervals.
  - **Surface Intersection:** Each face of the cube is checked for intersection with the isosurface by comparing intervals, ensuring that the resulting mesh correctly represents the possible positions of the isosurface within the given uncertainties.
  - **Error Propagation Control:** Interval arithmetic helps control error propagation in 3D mesh generation. When scalar values are uncertain, intervals ensure that the algorithm produces surfaces that cover all potential configurations.

Advantages and Disadvantages

- **Advantages:**
  - **Robustness:** Interval arithmetic increases the robustness of the algorithms by accounting for uncertainties and ensuring that all possible values are considered.
  - **Accuracy:** It improves the accuracy of the extracted contours or surfaces, making the algorithms suitable for applications requiring high precision.
  - **Error Management:** It provides a systematic way to manage and propagate errors through computations.
- **Disadvantages:**
  - **Performance Overhead:** The use of intervals can introduce computational overhead due to the need to perform operations on intervals rather than single values.
  - **Complexity:** Implementing interval arithmetic can add complexity to the algorithms, requiring careful handling of interval operations.

3.6 Pregunta 17: Practical Question: Texture Mapping

https://computergraphics.stackexchange.com/questions/1866/how-to-map-square-texture-to-triangle

Question: You are given a triangle with vertices at texture coordinates (0,0), (1,0), and (0,1). How would you assign texture coordinates to a point P within the triangle using barycentric coordinates? Provide a numerical example.

Answer:

Para asignar coordenadas de textura a un punto P dentro de un triángulo usando coordenadas baricéntricas, podemos seguir estos pasos:

1. **Definir las coordenadas de los vértices:**
   - Vértice A: (0,0)
   - Vértice B: (1,0)
   - Vértice C: (0,1)
2. **Calcular las coordenadas baricéntricas:**
   - Supongamos que el punto P tiene coordenadas (x, y) dentro del triángulo.
   - Las coordenadas baricéntricas (α, β, γ) de P en relación con los vértices A, B y C se pueden calcular utilizando la fórmula:

$$\alpha = \frac{(y_C - y_B) \cdot (x - x_B) + (x_B - x_C) \cdot (y - y_B)}{(y_C - y_B) \cdot (x_A - x_B) + (x_B - x_C) \cdot (y_A - y_B)}$$

$$\beta = \frac{(y_A - y_C) \cdot (x - x_C) + (x_C - x_A) \cdot (y - y_C)}{(y_A - y_C) \cdot (x_B - x_C) + (x_C - x_A) \cdot (y_B - y_C)}$$

$$\gamma = 1 - \alpha - \beta$$

3. **Asignar las coordenadas de textura:**

   - Usando las coordenadas baricéntricas, las coordenadas de textura (u, v) del punto P se calculan como:

$$u = \alpha \cdot u_A + \beta \cdot u_B + \gamma \cdot u_C$$

$$v = \alpha \cdot v_A + \beta \cdot v_B + \gamma \cdot v_C$$

   donde (u_A, v_A), (u_B, v_B) y (u_C, v_C) son las coordenadas de textura de los vértices A, B y C, respectivamente.

Ejemplo Numérico
Supongamos que el punto P tiene las coordenadas $(0.25, 0.25)$.

1. **Vértices del triángulo:**

   - $A = (0, 0)$
   - $B = (1, 0)$
   - $C = (0, 1)$

2. **Calcular las coordenadas baricéntricas:**

   - α:

$$\alpha = \frac{(1-0)\cdot(0.25-0)+(0-0)\cdot(0.25-0)}{(1-0)\cdot(0-0)+(0-0)\cdot(0-0)} = 0.25$$

   - β:

$$\beta = \frac{(0-1)\cdot(0.25-0)+(0-0)\cdot(0.25-1)}{(0-1)\cdot(1-0)+(0-0)\cdot(0-1)} = 0.25$$

   - γ:

$$\gamma = 1 - \alpha - \beta = 1 - 0.25 - 0.25 = 0.5$$

3. **Calcular las coordenadas de textura:**

   - u:

$$u = \alpha \cdot u_A + \beta \cdot u_B + \gamma \cdot u_C = 0.25 \cdot 0 + 0.25 \cdot 1 + 0.5 \cdot 0 = 0.25$$

   - v:

$$v = \alpha \cdot v_A + \beta \cdot v_B + \gamma \cdot v_C = 0.25 \cdot 0 + 0.25 \cdot 0 + 0.5 \cdot 1 = 0.5$$

Por lo tanto, las coordenadas de textura para el punto $P(0.25, 0.25)$ son $(u, v) = (0.25, 0.5)$.

3.7 Pregunta 19: Interval Arithmetic Application

Question: Consider the function $f(x) = x^2 - 2x + 1$. Use interval arithmetic to calculate the range of values of $f(x)$ when $x \in [1, 2]$. Explain the steps of the calculation.

Answer:

Para calcular el rango de valores de la función $f(x) = x^2 - 2x + 1$ cuando $x \in [1, 2]$ usando aritmética de intervalos, seguimos estos pasos:

Paso 1: Definir la función y el intervalo

La función dada es:

$$f(x) = x^2 - 2x + 1$$

El intervalo para x es:

$$x \in [1, 2]$$

Paso 2: Evaluar cada término de la función usando aritmética de intervalos

1. **Cuadrado del intervalo:**

$$x^2 \text{ cuando } x \in [1, 2]$$

   Para encontrar $[1, 2]^2$, calculamos los cuadrados de los extremos del intervalo:

$$[1, 2]^2 = [1^2, 2^2] = [1, 4]$$

2. **Producto del intervalo con -2:**

$$-2x \text{ cuando } x \in [1, 2]$$

Para encontrar $-2 \cdot [1, 2]$, multiplicamos cada extremo del intervalo por -2:

$$-2 \cdot [1, 2] = [-2 \cdot 2, -2 \cdot 1] = [-4, -2]$$

3. **Constante:** La constante en la función es 1, por lo que su intervalo es simplemente:

$$1 \in [1, 1]$$

Paso 3: Sumar los intervalos obtenidos
Ahora sumamos los intervalos que obtuvimos de cada término:

1. **Sumar x^2 y $-2x$:**

$$[1, 4] + [-4, -2] = [1 + (-4), 4 + (-2)] = [-3, 2]$$

2. **Sumar el resultado anterior con la constante:**

$$[-3, 2] + [1, 1] = [-3 + 1, 2 + 1] = [-2, 3]$$

Por lo tanto, el rango de valores de $f(x) = x^2 - 2x + 1$ cuando $x \in [1, 2]$ es $[-2, 3]$.