Examen Final Computacion Grafica

Jeremy Matos Cangalaya

July 8, 2024

Contents

1 Pregunta 1

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

1.1 Implicit Representation

Define una superficie como un conjunto de puntos x, y, z que satisfacen la ecuación matématica $f(x, y, z) = 0$. Por ejemplo, la esfera se puede definir como $x^2 + y^2 + z^2 - r^2 = 0$. La ventaja de esta representación es que es fácil de definir y de manipular. Sin embargo, la desventaja es que no es útil para representar intersecciones con rayos o geometrías compuestas y complejas.

1.2 Explicit Representation

Define una superficie a partir de una función matemática que asigna un valor de z a cada par de coordenadas (x, y). Por ejemplo, la función $z = x^2 + y^2$ define una paraboloide circular. La ventaja de esta representación es que es útil para el cálculo de intersecciones, diferencias. Sin embargo, la desventaja es que geometrías que no pueden ser representadas de la forma $f(x, y)$ como el toroide no pueden ser representadas de forma explícita.

1.3 Parametric Representation

Define la superficie como una función de dos parámetros u, v de la forma $x = f(u, v), y = g(u, v), z = h(u, v)$. Por ejemplo, la esfera se puede definir como $x = r\cos(u)\sin(v), y = r\sin(u)\sin(v), z = r\cos(v)$. La ventaja de esta representación es que es útil para representar geometrías complejas y compuestas. Sin embargo, la desventaja es que es más difícil de definir y manipular que las representaciones explícitas e implícitas.

2 Pregunta 2

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

Respuesta

El *radiosity algorithm* se considera como *viewpoint-independent* porque el cálculo de la iluminación se realiza en función de la interacción entre las superficies de la escena y no en función de la posición del observador. El algoritmo de radiosity se basa en el cálculo de la transferencia de energía lumínica entre las superficies de la escena. El proceso de renderizado usando el algoritmo de radiosity se puede dividir en los siguientes pasos:

- **Discretización de la escena:** La escena se divide en parches o elementos de superficie que se utilizan para calcular la transferencia de energía lumínica.
- **Cálculo de la matriz de forma de onda de energía:** Se calcula la matriz de forma de onda de energía que describe la transferencia de energía lumínica entre los parches de la escena.
- **Resolución del sistema de ecuaciones:** Se resuelve el sistema de ecuaciones lineales que describe la transferencia de energía lumínica entre los parches de la escena.
- **Cálculo de la iluminación:** Se calcula la iluminación de la escena en función de la transferencia de energía lumínica entre los parches de la escena.
- **Renderizado de la escena:** Se renderiza la escena utilizando la iluminación calculada.

3 Pregunta 3

Describe the half-edge represnetation for meshes. Propose implementations (as efficient as possible) for the following methods, given a half-edge representation of a mesh. Analyze the complexity of the proposed method.

3.1 Half-edge representation

Descripción: La representación de half-edge es una estructura de datos utilizada para representar mallas poligonales.

Implementación

- **Half-edge**: Cada arista es dividida en dos half-edges, una para cada cara adyacente. Cada half-edge almacena referencias a la arista, la cara adyacente, la half-edge siguiente y la half-edge gemela.

- **Vertex**: Almacena referencias a las half-edges que tienen el vértice como origen.
- **Face**: Almacena referencias a las half-edges que forman la cara.
- Iterate through the neighbors of a vertex: Traverse all vertices connected to a given vertex

```
    def neighbors(vertex):  # time complexity: O(n)
        neighbors = []
        edge = vertex.edge
        start = edge
        while True:
            neighbors.append(edge.vertex)
            edge = edge.twin.next   # Next half-edge
            if edge == start:
                break
        return neighbors

```

- Collapse an edge: reduce and edge to a single vertex, simplifying the mesh by removing vertices and faces

```

    def collapse_edge(edge):    # time complexity: O(n)
        v1, v2 = edge.vertex, edge.twin.vertex
        f1, f2 = edge.face, edge.twin.face

        # reasignar la posicion del vertice a la mitad
        v1.position = (v1.position + v2.position) / 2

        # reasignar las aristas
        for e in f2.edges:
            e.face = f1

        # remover aristas y caras
        f1.mesh.faces.remove(f2)
        f1.mesh.edges.remove(edge)
        f1.mesh.edges.remove(edge.twin)
        f1.mesh.vertices.remove(v2)


```

- Find boundary edges: Identify edges that are only connected to one face, which delinearte the boundary of the mesh

```

    def boundary_edges(mesh):   # time complexity: O(n)
        boundary_edges = []
        for edge in mesh.edges:
            if edge.twin is None:
                boundary_edges.append(edge)
        return boundary_edges

```

4 Pregunta 4

- Describe briefly the Painter's Algorithm for scene rendering. What are its limitations? **Painter's algorithm:** El algoritmo del pintor es un algoritmo de renderizado utilizado en gráficos 3D para determinar el orden en que se deben dibujar los objetos en la escena. El algoritmo se basa en la idea de que los objetos más cercanos al observador deben ser dibujados antes que los objetos más lejanos. El algoritmo del pintor se puede resumir en los siguientes pasos:
  - Calcular la profundidad de cada objeto en la escena.
  - Ordenar los objetos en función de su profundidad.
  - Pintar los objetos en el orden determinado.
  - Actualizar el búfer de profundidad para evitar que los objetos se superpongan incorrectamente.
  - Repetir el proceso para cada fotograma de la animación.
- Analyze very carefully the situation in which there are intersecting triangles as input in the Painter's algorithm (for example: situations in which the triangles intersect in the space and it is not possible to obtain a correct image by just defining th order in which we paint them, because any ordering will be incorrect). These cases require to split the triangles somehow. Design and implement a method for doing so. Your method should be abble to draw things like the following image:

Respuesta: Para abordar el problema de los triangulos superpuestos del algoritmo del pintor, se puede dividir los triangulos en subtriangulos que no se superpongan. El detalle se encuentra en preservar el orden de los triangulos subdivididos para que se dibujen correctamente, esto se logra, verificando en cada retriangulacion los triangulos que no tienen intersecciones. Si un triangulo luego de una retriangulacion tiene intersecciones, se vuelve a subdividir hasta que todos los triangulos no se superpongan. El algoritmo se puede implementar de la siguiente manera:

```
    def calculate_triangle_order(triangles, depth = 1):
        ordered_triangles, other_ordered_triangles = [], []
        for triangle in triangles:
            if triangle.is_overlapping():
                smaller_triangles = split_triangle(triangle)
                other_ordered_triangles.extend(calculate_triangle_order(smaller_triangles, depth + 1))
            else:
                ordered_triangles.append(triangle)
        other_ordered_triangles = sorted(other_ordered_triangles, key=lambda t: t.depth, t.area)
        ordered_triangles.extend(other_ordered_triangles)
        return ordered_triangles

```

5 Pregunta 5

Given a triangle $T_1 = A_1B_1C_1$ and a triangle $T_2 = A_2B_2C_2$ and given a point $p \in T_1$, such that $p_1 = (p_{1_x}, p_{1_y})$ and $A_1 = (A_{1_x}, A_{1_y})$, etc. , propose a method to compute the coordinates of the point $p_2 \in T_2$ with the same baricentric coordinates in T_2 as p_1 in T_1. This is: porpose a method to implement a map of T_1 into T_2 using baricentric coordinates.

1. Calcular las coordenadas baricéntricas de p_1 en T_1:

   Sea $p_1 = (p_{1x}, p_{1y})$ y los vértices $A_1 = (A_{1_x}, A_{1_y})$, $B_1 = (B_{1_x}, B_{1_y})$, $C_1 = (C_{1_x}, C_{1_y})$.

   Resolver el sistema de ecuaciones:

$$\begin{cases} A_{1_x}\alpha + B_{1_x}\beta + C_{1_x}\gamma = p_{1_x} \\ A_{1_y}\alpha + B_{1_y}\beta + C_{1_y}\gamma = p_{1_y} \\ \alpha + \beta + \gamma = 1 \end{cases}$$

2. Usar las coordenadas baricéntricas (α, β, γ) para encontrar p_2 en T_2:

   Sea $A_2 = (A_{2_x}, A_{2_y})$, $B_2 = (B_{2_x}, B_{2_y})$, $C_2 = (C_{2_x}, C_{2_y})$.

   Las coordenadas de p_2 se calculan como:

$$p_{2_x} = \alpha A_{2_x} + \beta B_{2_x} + \gamma C_{2_x}$$

$$p_{2_y} = \alpha A_{2_y} + \beta B_{2_y} + \gamma C_{2_y}$$

De este modo se puede obtener un mapeo de un punto p_1 en un triángulo T_1 a un punto p_2 en un triángulo T_2 usando coordenadas baricéntricas.

6 Pregunta 6

Develop an algorithm for calculating the intersection between a ray and a sphere in a ray tracing system. Explain each step of the pseudocode. Inputs origin and direction of the ray, center of sphere, radius of the sphere.

Respuesta

Ray-Sphere Intersection

Para calcular la intereseccion entre un rayo y una esfera tenemos que tener en cuenta que el rayo atraviesa 2 veces la esfera, por lo que podemos calcular la distancia de los puntos de intersección y seleccionar el más cercano al origen del rayo o hacer el calculo entre ambos para esto se propone el siguiente pseudocodigo:

- Triangularizar la esfera con un algoritmo de mesh simplification para obtener un polígono que aproxime la esfera.
- Para cada triangulo realizar en paralelo el algoritmo de intersección rayo-triángulo de Möller-Trumbore explicado a continuacion.

Ray-Triangle Intersection

1. Dado un rayo definido por un punto de origen O y un vector de dirección D, y un triángulo definido por tres vértices V_0, V_1 y V_2.
2. Calcula el vector normal N del triángulo tomando el producto cruz de los vectores $\overrightarrow{V_1V_0}$ y $\overrightarrow{V_2V_0}$.
3. Calcula el producto punto del vector de dirección del rayo D y el vector normal del triángulo N. Si el producto punto es cercano a cero, el rayo es paralelo al triángulo y no hay intersección.
4. Calcula la distancia t desde el origen del rayo O hasta el punto de intersección usando la fórmula:

$$t = \frac{(\overrightarrow{V_0O} \cdot N)}{(D \cdot N)}$$

5. Si t es negativo, el punto de intersección está detrás del origen del rayo y no hay intersección.
6. Calcula el punto de intersección P usando la fórmula:

$$P = O + tD$$

7. Calcula las coordenadas baricéntricas (u, v) del punto de intersección P con respecto a los vértices del triángulo V_0, V_1 y V_2.
8. Si $u \geq 0$, $v \geq 0$ y $u + v \leq 1$, el punto de intersección P está dentro del triángulo y hay una intersección.
9. Devuelve el punto de intersección P y las coordenadas baricéntricas (u, v).

```python
def ray_triangle_intersection(ray_origin, ray_direction, triangle_vertices):
    v0, v1, v2 = triangle_vertices
    n = cross_product(v1 - v0, v2 - v0)

    if np.dot(ray_direction, n) < eps:    # check if parallel
        return None

    # calculate distance to intersection point
    t = np.dot(v0 - ray_origin, n) / np.dot(ray_direction, n)

    # check if intersection point is behind the ray origin
    if t < 0:
        return None

    # calculate intersection point
    intersection_point = ray_origin + t * ray_direction

    # calculate barycentric coordinates
    u, v = calculate_barycentric_coordinates(intersection_point, v0, v1, v2)

    # check if intersection point is inside the triangle
    if u >= 0 and v >= 0 and u + v <= 1:
        return intersection_point, (u, v)
    else:
        return None
```

Este algoritmo es conocido como el algoritmo de intersección rayo-triángulo de Möller-Trumbore y es ampliamente utilizado en motores de renderizado y trazadores de rayos.

7 Pregunta 7

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.

7.1 Respuesta

- **Paso 1:** Definir el intervalo $x \in [1, 2]$
- **Paso 2:** Calcular el cuadrado del intervalo $x^2 = [1, 4]$
- **Paso 3:** Calcular el producto de $2x = 2[1, 2] = [2, 4]$
- **Paso 4:** Transformar el 1 a un intervalo $1 = [1, 1]$

- **Paso 5:** Calcular la resta de los intervalos $x^2 - 2x = [1, 4] - [2, 4] = [-3, 2]$
- **Paso 6:** Calcular la suma de los intervalos $x^2 - 2x + 1 = [-3, 2] + [1, 1] = [-2, 3]$

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

8 Pregunta 8

Design a pinhole camera system to capture an image of a building that is 10 meters tall and located 15 meters away. If the camera has a 24 mm high sensor, and we want the building to fill 90-100% of the height of the sensor, what should be the focal length? Perform the necessary calculations and justify your answers.

8.1 Answer

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

$$\frac{f}{d} = \frac{h}{H}$$

Donde:

- f: es la distancia focal de la cámara
- d: es la distancia al objeto = 15 metros o 15000 mm
- H: altura del objeto en la escena = 10 metros o 10000 mm
- h: altura del sensor = 24 mm, y queremos que el objeto ocupe entre el 90% y el 100% de la altura del sensor. Por lo tanto:
  - Para el 90%: $h = 0.9 \times 24 = 21.6\,\text{mm}$
  - Para el 100%: $h = 24.0\,\text{mm}$

Calculamos la distancia focal para ambos casos:

$$f_{90\%} = \frac{d \times h_{90\%}}{H} = \frac{15000 \times 21.6}{10000} = 32.4\,\text{mm}$$

$$f_{100\%} = \frac{d \times h_{100\%}}{H} = \frac{15000 \times 24}{10000} = 36.0\,\text{mm}$$

Por lo tanto, la distancia focal de la cámara debe estar entre aproximadamente 32.4 mm y 36.0 mm para capturar la imagen del edificio llenando entre el 90% y el 100% de la altura del sensor.

9 Pregunta 9

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

9.1 SIFT

Keypoint extraction

- **Detección de puntos de interés en el espacio de escala:** Detecta posibles puntos de interés en la imagen buscando máximos y mínimos locales en la función de diferencia de Gaussianas.
- **Localización de puntos clave:** Refina la ubicación de los puntos clave ajustando una función cuadrática al vecindario local de los máximos y mínimos.
- **Asignación de orientación:** Asigna una orientación a cada punto clave basándose en la magnitud y orientación del gradiente de la imagen.
- **Descriptor de puntos clave:** Calcula un descriptor para cada punto clave basándose en la magnitud y orientación del gradiente de la imagen.

Keypoint Matching

- **Descriptor Matching**: Se usa la distancia euclideana como medida de similitud entre los descriptores.
- **Lowe's Ratio Test**: Se descartan los emparejamientos de acuerdo a un threshold reduciendo la probabilidad de los falsos matches.

9.2 Harris Corners

Harris Corner Extraction

- **Gradient Calculation**: Calcula el gradiente de la imagen usando un filtro de Sobel o alguno similar.
- **Structure Tensor**: Calcula la matriz de covarianza de los gradientes en una vecindad local.
- **Corner Response Function**: Calcula la respuesta de esquina usando la matriz de covarianza.
- **Non-maximum Suppression**: Suprime los puntos que no son máximos locales.

Corner Matching

- **Descriptor Calculation**: Calcula un descriptor para cada esquina basado en la intensidad de los píxeles en la vecindad local o algun metodo como SIFT o SURF.
- **Descriptor Matching**: Se usa la distancia euclideana como medida de similitud entre los descriptores y del mismo modo que SIFT usa el **Lowe's Ratio Test** para descartar los emparejamientos de acuerdo a un threshold.