# Bitácora Taller 3 – Superficies Paramétricas y Bump Mapping

**Curso**: Computación Gráfica
**Fecha**: 3 de Septiembre 2025
**Estudiantes**: Abel Albuez Sanchez, Ricardo Crus
**Profesor**: Leonardo Florez-Valencia

---

Cuando empezamos este taller, teníamos una idea bastante abstracta: *hacer que una superficie plana pareciera tener relieve sin cambiar su geometría*. Sabíamos que el camino era implementar **bump mapping**, pero al principio no teníamos muy claro cómo lograr que una simple imagen en formato `.ppm` se convirtiera en un efecto 3D convincente.

El reto era doble: por un lado, entender las **superficies paramétricas** y cómo parametrizarlas en `(u,v)` para generar diferentes geometrías; y por otro, aplicar el concepto de **perturbación de normales** para simular textura y volumen. El taller no fue solo escribir código, sino ir descubriendo paso a paso qué funcionaba, qué fallaba, y cómo cada ajuste nos acercaba a un resultado más realista.

---

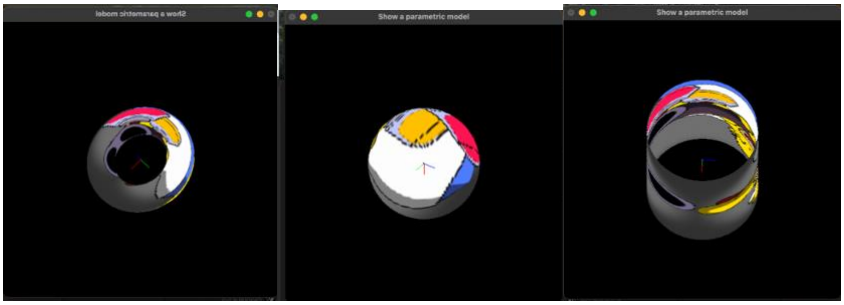
# Etapa 1 – Superficies paramétricas

Comenzamos definiendo la función `parametric_model(point, normal, u, v)` como la base de todo. Al inicio, `(u,v)` estaban en el rango `[-0.5, 0.5]`, lo cual funcionaba bien para el plano, pero cuando quisimos pasar a superficies como la esfera o el toroide nos dimos cuenta de que **ese rango no alcanzaba para recorrer ángulos completos**. Entonces agregamos un `remap_to_2pi` que convertía `(u,v)` al rango `[0, 2π]`. Ese pequeño cambio en el código fue lo que nos abrió la puerta a las superficies circulares.

Probamos con varias fórmulas:

- **Plano:** `point[2] = 0`. Fue nuestro caso base, útil para validar normales e iluminación.
- **Cilindro:** usamos `x = cos(θ), y = sin(θ), z = altura`. Aquí aprendimos que había que escalar `v` para controlar la altura.
- **Esfera:** implementamos coordenadas esféricas con `θ` y `φ`, donde `φ` iba de `0` a `π`. El problema fue que en los polos las normales se colapsaban.
- **Elipsoide:** modificamos la esfera con radios distintos `rx, ry, rz`. Ese simple cambio de escala nos enseñó cómo deformar superficies paramétricas.
- **Toroide:** aplicamos la fórmula `(R + r·cosφ)·cosθ, (R + r·cosφ)·sinθ, r·sinφ`. Fue la más estable porque no tenía polos con singularidades.

Aunque en el **código final solo dejamos el plano**, todas estas pruebas nos enseñaron cómo los **rangos y ángulos afectan la geometría y el mapeo UV**.



---

# Etapa 2 – Fundamentos de iluminación

Al principio pensábamos que el bump mapping ya estaba funcionando porque movíamos las normales en el código… pero en pantalla no pasaba nada. La clave fue descubrir que no habíamos activado la iluminación. En `App::init()` agregamos:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

y configuramos la luz con componentes ambiente, difusa y especular.

Ese cambio fue un punto de quiebre: de repente, la misma superficie que antes se veía gris y plana ahora reaccionaba a la luz. Lo interesante es que con normales constantes (`0,0,1`) veíamos una iluminación uniforme, pero si inclinábamos las normales manualmente, la superficie empezaba a reflejar diferente.

Aquí aprendimos que el **bump mapping no existe sin iluminación**, porque son las normales las que controlan cómo la luz rebota. El cambio en el código no fue mucho, pero el cambio en la visualización fue enorme.

---

# Etapa 3 – Primer bump mapping

Luego integramos la imagen `.ppm`. En `drawBumpMappedSurface` lo primero fue tomar la **intensidad de un píxel como altura**:

```
float h = pixel[0] / 255.0f;
```

```
normal[0] += (h - 0.5f) * bumpStrength;
normal[1] += (h - 0.5f) * bumpStrength;
```

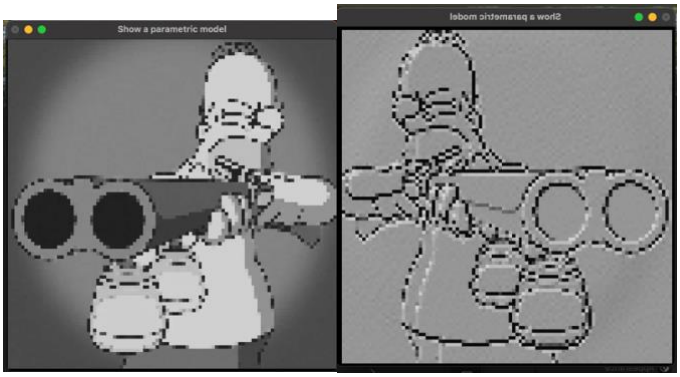
Eso nos dio un relieve visible, pero demasiado uniforme, como si toda la superficie estuviera inflada.

Entonces hicimos el cambio grande: **usar gradientes**. Calculamos diferencias con los píxeles vecinos derecho y superior:

```
float dx = (hRight - h) * bumpStrength;
float dy = (hUp - h) * bumpStrength;
normal[0] -= dx;
normal[1] -= dy;
```

Esto fue un salto de calidad: ahora los bordes de la imagen aparecían como cambios de relieve y el plano empezaba a “tener volumen” sin que la geometría cambiara.

En esta etapa aprendimos que **el valor absoluto de un píxel no es tan importante como la diferencia con sus vecinos**, porque eso es lo que genera la ilusión de relieve.




---

# Etapa 4 – Suavizado e interactividad

Cuando subimos el `gridSize` de 50 a 100 para ganar detalle, apareció un problema: mucho **ruido visual**. Cada píxel de la imagen generaba un salto brusco en la normal.

La solución fue agregar `sampleImageSmooth`, que promedia un bloque de 3x3 píxeles. Ese cambio en el código redujo los artefactos y nos permitió activar o desactivar el suavizado con la tecla `S`.

Además, hicimos el sistema interactivo con `_cb_keyboard`:

- `+ / -` para ajustar la fuerza (`m_bumpStrength`).
- `S` para suavizado.
- `L` para rotar la luz (`m_lightAngle`).
- `R` para resetear la cámara.

Aquí aprendimos que **un algoritmo no basta, hay que hacerlo controlable**, porque solo experimentando con distintos valores entendimos realmente cómo afectaban al resultado.

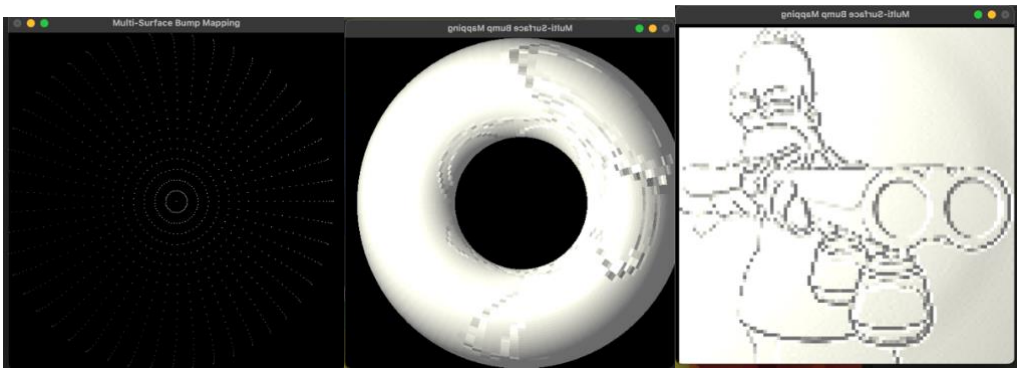
---

# Etapa 5 – Multi-superficies (fase experimental)

Finalmente aplicamos todo lo anterior a otras superficies dentro de `parametric_model`. Ajustamos las fórmulas de esfera, cilindro y toroide para ver cómo se comportaba el bump mapping.

- En la **esfera**, los polos producían distorsión: la textura se estiraba y el efecto se rompía.
- En el **cilindro**, se notaban las costuras al cerrar el ángulo.
- En el **toroide**, en cambio, el mapeo fue más estable y el bump se distribuía de manera uniforme.

Aunque en el código final **solo dejamos el plano**, esta etapa fue clave para entender que **la topología de la superficie influye directamente en la calidad del bump mapping**.




---

# Lo que aprendimos

Este taller nos mostró que el verdadero reto no era solo escribir código, sino **entender cómo cada decisión matemática y visual impacta en el resultado final**. Al pasar de rangos simples a parametrizaciones completas, comprobamos que la geometría no es un detalle secundario: la forma en que definimos el espacio condiciona la calidad de cualquier técnica posterior, incluyendo el bump mapping.

La iluminación fue otro descubrimiento fundamental. Nos dimos cuenta de que las normales por sí solas no significan nada si no existe una fuente de luz que revele sus diferencias. Esta etapa nos enseñó que, en gráficos por computador, los datos necesitan siempre un contexto visual para cobrar sentido. La luz fue ese puente entre lo que calculábamos en el código y lo que realmente se podía percibir en pantalla.

Con el bump mapping aprendimos a valorar los **pequeños cambios locales**. Un solo píxel no dice mucho, pero al compararlo con sus vecinos aparecen gradientes que transforman un plano estático en una superficie llena de matices. La adición de suavizado y controles interactivos reforzó esa idea: el detalle y la experimentación en tiempo real son claves para pasar de un resultado académico a una experiencia visual convincente.

Finalmente, el paso por múltiples superficies nos dejó una lección más amplia: el efecto no se comporta igual en todas las geometrías. Un toroide puede lucir impecable mientras una esfera exhibe sus limitaciones en los polos. Esa experiencia nos recordó que **la técnica nunca vive aislada**: depende de la topología, de la parametrización y de cómo decidimos implementarla. Al concluir, entendimos que el bump mapping es una ilusión poderosa, y que detrás de cada relieve visible hay un proceso lleno de ajustes, pruebas y aprendizajes.