# PART A

Το Terrain δημιουργήθηκε με το gaea και ύστερα φορτώθηκε από την OpenGL (.obj αρχείο).

Έγινε χρήση κάποιων βασικών textures στην αρχή για τη λίμνη και το έδαφος. Ο διαχωρισμός ανάμεσα στα δύο πραγματοποιήθηκε με ένα lake mask

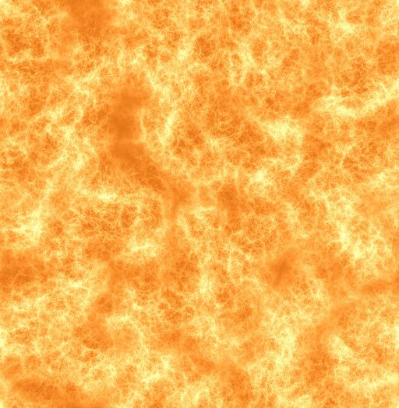
\_Kd.rgb = mix(terrainColor, finalWater, lakeMask);

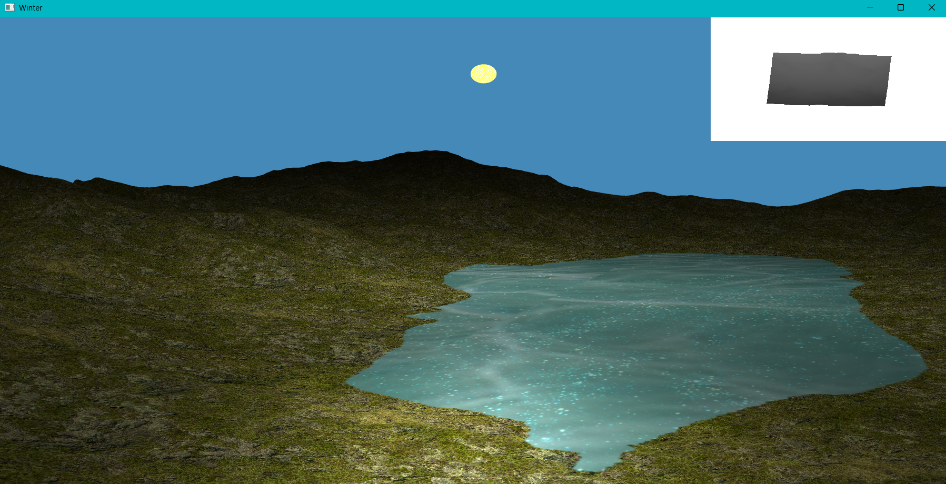
Σημείωση: Έχει χρησιμοποιηθεί μοντέλο Phong.

Για την εφαρμογή υφών σε άλλα αντικείμενα, τροποποιήθηκε η λογική του shader ως εξής:

uniform int useTexture = 0;

1. Απλά υλικά: (mtl struct)
2. Υφή terrain (λίμνη και έδαφος)
3. Άλλα αντικείμενα με υφή όπως Sun Sphere Texture



**Αρχική Εμφάνιση:**

## Φωτεινή πηγή



## Ουρανός/Σύννεφα

Αρχικά έθεσα το background χρώμα του καρέ σε γαλάζιο, όμως για πιο ρεαλιστική εμφάνιση δημιουργήθηκε skydome

Skydome

**Positioning and the Camera 🔭:** To make the skydome appear infinitely far away and prevent the user from "running into" its walls, we need to ensure the sphere's world translation (its model matrix's translation component) is always the **same as the camera's position**. This keeps the camera at the center of the sphere, no matter where it moves, making the dome feel static and distant.

**Depth and Drawing Order 🧊:** The skydome should typically be the **first** thing drawn in the scene, and you might want to adjust the depth check (GL\_DEPTH\_FUNC) or the depth value of the sky fragments to ensure other objects (the terrain, the sphere) appear in front of it.



Vertex shader

vertex\_UV = spherical\_mapping(vertexPosition);

 Υπολόγιζες **σωστό UV σε κάθε κορυφή**

 Μετά το GPU έκανε **γραμμική παρεμβολή (linear interpolation)**:

vertex\_UV ------> fragment\_UV

Η γωνία atan() **ΔΕΝ είναι γραμμικό μέγεθος**.  
Όταν ένα τρίγωνο περνάει από το σημείο:

U = 0.99 ----- seam ----- U = 0.01

το GPU νομίζει ότι πρέπει να πάει:

0.99 → 0.01 = μεγάλη διαδρομή προς τα ΠΙΣΩ

αντί για τη σωστή:

0.99 → 1.00 → 0.00 → 0.01 = μικρή διαδρομή ΓΥΡΩ ΑΠΟ ΤΗ ΣΦΑΙΡΑ

📌 Άρα:

✅ Μαθηματικά τα UV στα vertices ήταν σωστά  
❌ ΑΛΛΑ η **παρεμβολή τους ήταν λάθος λόγω κυκλικής γεωμετρίας**

# ΤΙ ΑΛΛΑΞΑΜΕ ΤΩΡΑ (Fragment Shader)

Τώρα κάνεις αυτό:

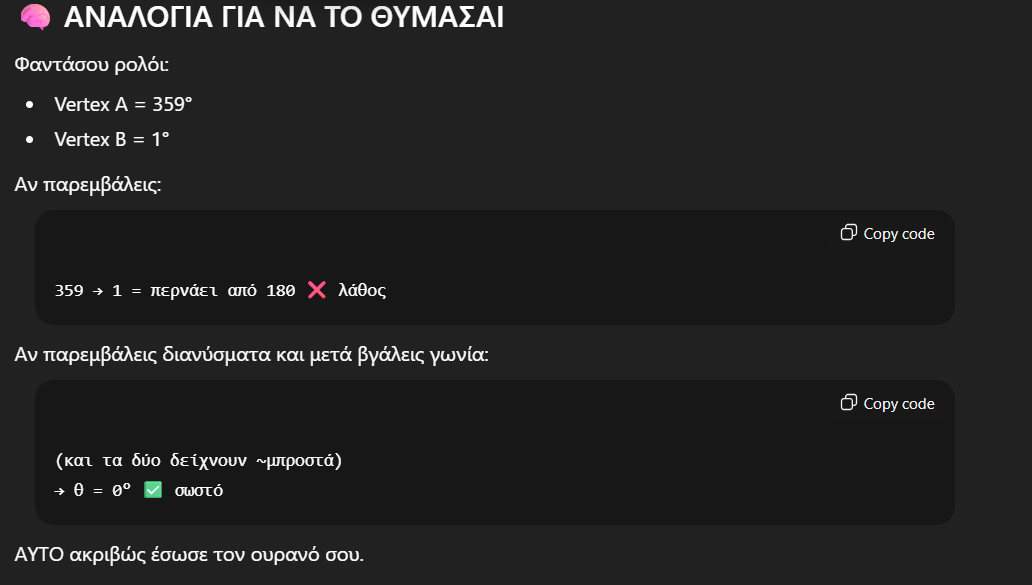
vec3 p = normalize(fragPos\_modelspace);

u = atan(p.z, p.x);

v = asin(p.y);

Δηλαδή:

✅ Δεν παρεμβάλεις πλέον u,v  
✅ Παρεμβάλεις **ΜΟΝΟ καρτεσιανή θέση (x,y,z)**  
✅ Και υπολογίζεις την πολική μετατροπή ΑΝΑ PIXEL



**✅ ΤΙ ΠΡΕΠΕΙ ΝΑ ΘΥΜΑΣΑΙ ΓΙΑ ΠΑΝΤΑ**

✅ Πάντα:

* Παρεμβάλουμε **γραμμικά μεγέθη** (θέσεις, κατευθύνσεις)
* **Όχι γωνίες, όχι σφαίρες, όχι atan, όχι polar coordinates στο vertex shader**

✅ Οι μη-γραμμικές μετατροπές:

* atan
* asin
* normalize
* length

👉 Καλύτερα να γίνονται **στο fragment shader όταν υπάρχει πιθανότητα seam**

## Ρεαλιστική Κάμερα – Παίχτης στο heightmap

#define SCALING\_FACTOR 60 στο lab.cpp kai camera.cpp

## C:\Users\USER\AppData\Local\Temp\{7B15E160-4123-44D2-BF2A-56AA1B525DDA}.tmp.