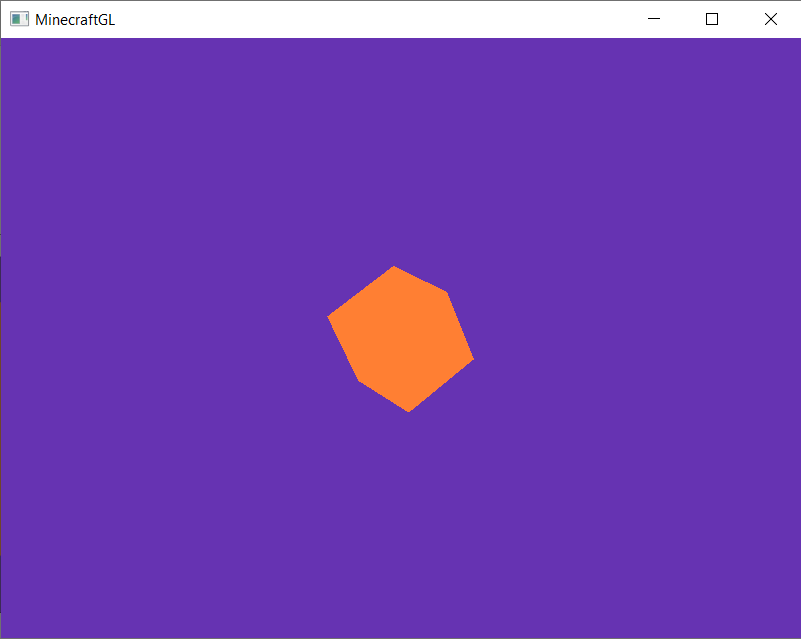
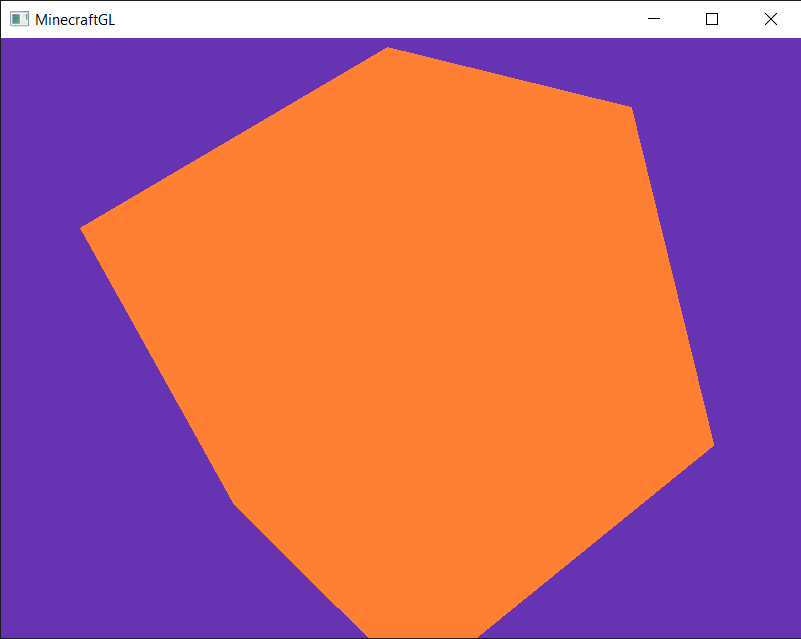
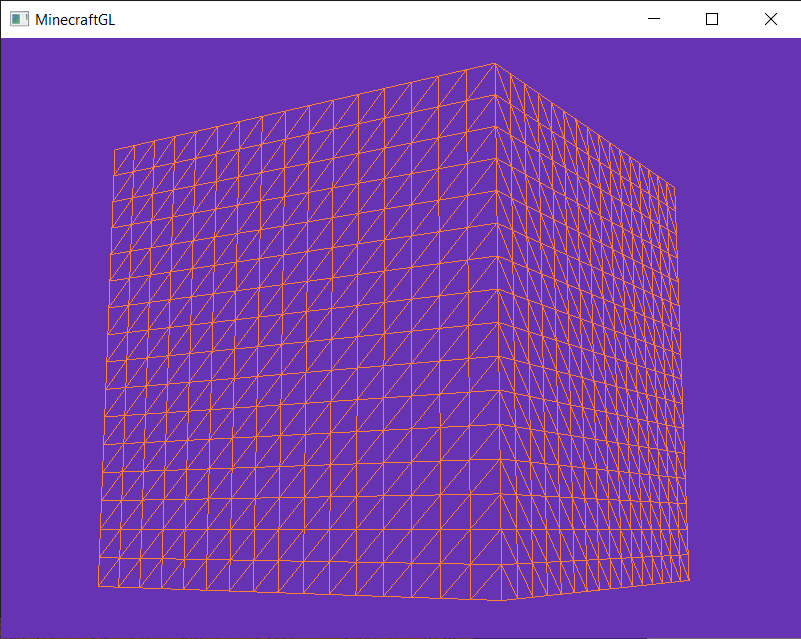
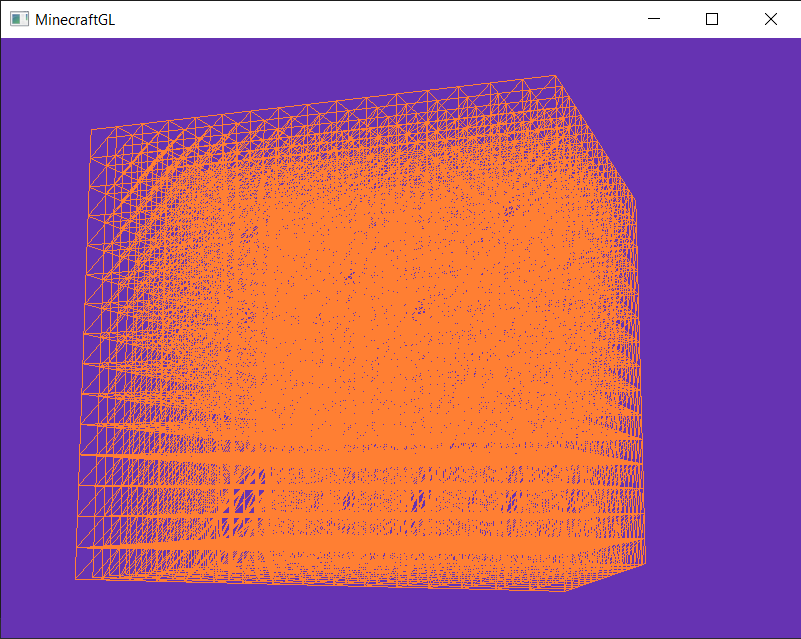
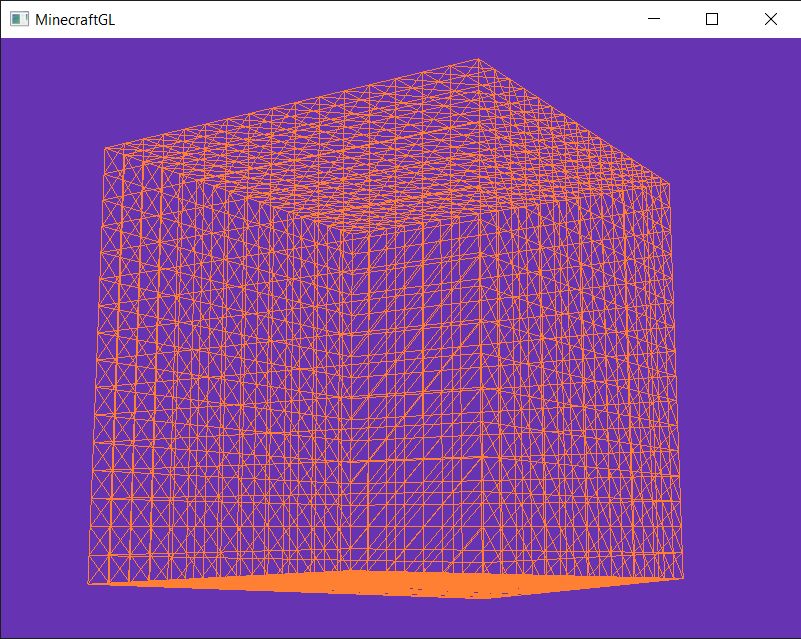
****

****

****

**Cub**

Un **cub** és el sinònim de *voxel*. El món està format per una gran multitud de cubs. Cada cub té un tipus determinat: aire, terra, gespa, aigua... El jugador pot col·locar qualsevol cub que disposi al seu inventari i modificar el món. El tipus es guarda en un byte, de manera que obtenir la textura sigui més senzill.

**Chunk**

Un **chunk** és un conjunt de cubs. El món està format per un conjunt de chunks, i aquests seran els responsables de gestionar els cubs del món. Podem definir la mida que vulguem pels chunks amb **width**, **height** i **depth**, però per mantenir-los senzills farem que siguin de 16\*16\*255.

[HA CAMNIADO???]

Els cubs es guarden en una array tridimensional que representa el chunk. **No ens guardem els cubs com a tal ni la seva posició en una classe, sinó que ens guardem el tipus d’un cub**. D’aquesta manera sempre es renderitzaran 16\*16\*16 cubs (els cubs d’aire, per exemple, no es renderitzen, però segueixen existint a l’array). Com que hem dit que el tipus d’un cub és un byte, el tipus de l’array tridimensional serà *uint8\_t*.

Optimització

Pos: 2 bytes <- 3 bytes

Pos text: 1 byte <- 2 bytes

Tipo text: 1 byte <- 2 bytes

Llum: 1 byte

Podriem fer servir 5 bytes en comptes de 8. 983040 <<< 1572864, ens podríem estalviar un 62.5% d’espai en memòria.

// Posicio

vertices.push\_back(x);

vertices.push\_back(y);

vertices.push\_back(z);

// Color

//vertices.push\_back(r);

//vertices.push\_back(g);

//vertices.push\_back(b);

// Quantitat de llum

vertices.push\_back(llum);

// Coordenades de textura

vertices.push\_back(u);

vertices.push\_back(v);

// Tipus (posició del mapa de textures)

vertices.push\_back(tipus%16);

vertices.push\_back(tipus/16);

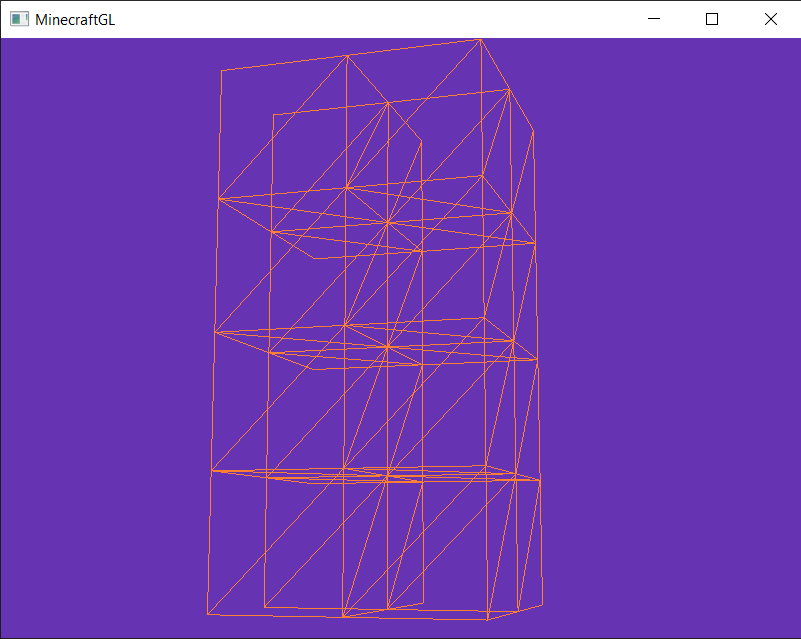
AHORA SI SE GUARDAN COMO CLASES, PROBAR A VER CUAL ES MEJOR??

Cada chunk té el seu propi VBO on col·locarà els vèrtexs de cada un dels cubs de l’array.

Actualitzar el chunk: significa crear els vèrtexs necessaris per renderitzar després el chunk. Els vèrtexs, en comptes de ser *floats*, son *uint8\_t* perquè **no pot haver-hi un cub tallat per la meitat**. No ens fa falta tanta informació, així que només utilitzem 1 byte per vèrtex (notem que si utilitzéssim floats, estaríem utilitzant 4 bytes per vèrtex). Hi ha dues maneres si es canvia algun cub del chunk: actualitzar tot el chunk o actualitzar només una part.

PROBAR CUAL ES MEJOR!!!!!

Ctime, srand y rand()



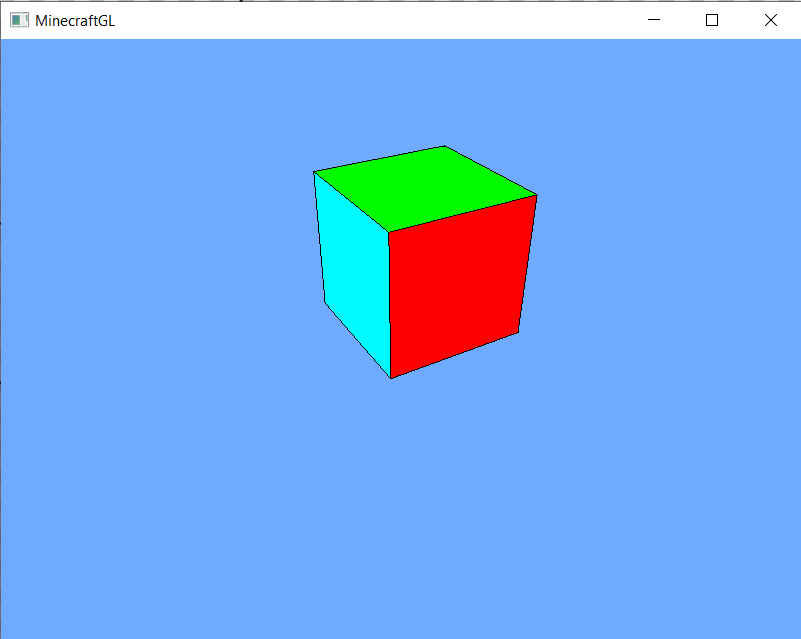
Tot i utilitzar el mode de culling de l’OpenGL, no s’eliminen les cares pegades a altres cubs.

Maneres d’optimització: culling

Juntar dos caras en una sola para no renderizar 4 triangulos para dos caras, sino dos triángulos muy largos: NO FUNCIONARIA?

Estamos utilitzando GLM 0.9.5 porque las nuevas no compilan en C++ 11.

“Picking”

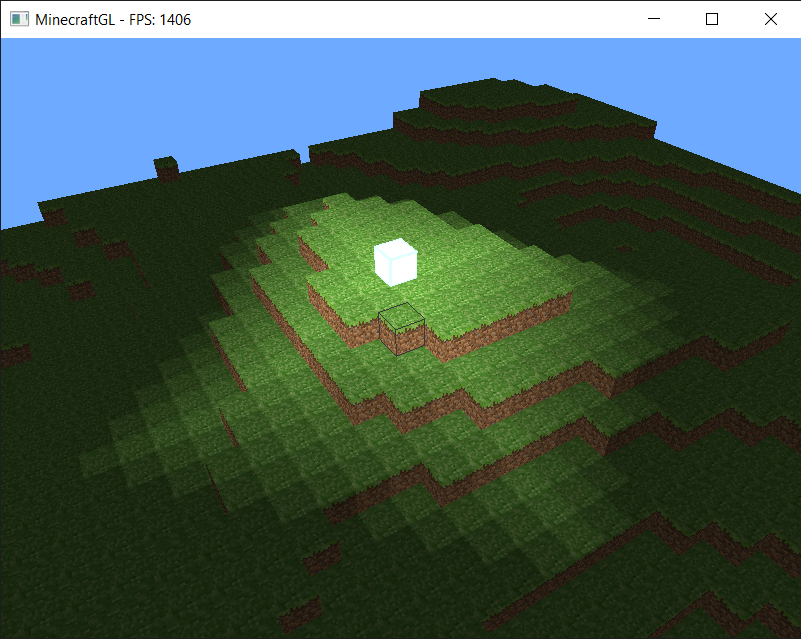


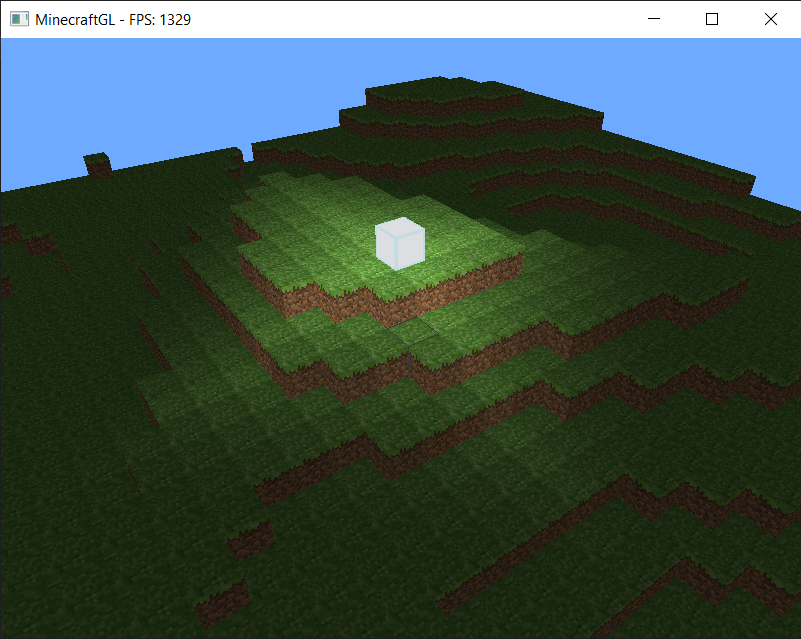
Iluminació

En un byte, 4 bits naturals, 4 bits artificials. Breadth-first search?

Avantatges: només s’ha de calcular un cop si posem una llum, no hem de fer servir shadow maps per cada llum que hi hagi. Ens faria falta global illumination. Quantes més llums, més lent aniria.

Correcció de gamma: 1.0, 0.5 i 2.5





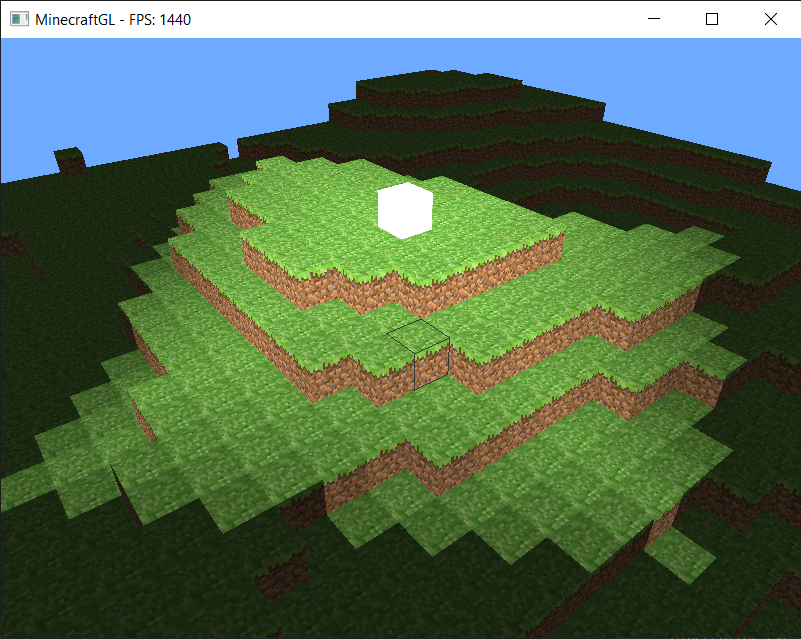


Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza media

* Sistema de chunks: probar diferentes maneras de gestionar cubos y mirar cuál es mejor, ventajas y desventajas
* Menu pausa
* Fullscreen, posibilidad de cambiar entre monitores
* Obtener recursos a partir de cubos, colisiones
* Mundo generado a partir de una seed: procedural
* Iluminación, lluvia (shaders)
* Ciclo día / noche: estrellas
* Montañas
* Árboles / estructuras
* Vegetación: flores, hierba...
* Nubes
* Sonido
* Ríos, lagos, mar
* Fuego
* Jugador: movimiento, inventario (posibilidad de cofres), poner bloques, quitarlos, sistema de crafteo, hambre y comida
* NPCs: animales (vacas, gallinas, ovejas), enemigos (...)
* Guardar partida, cargar partida
* Opcional: multijugador
* …