# Bistaratzea eta Ingurune Birtualak

2.Praktikaren entrega

Fernando Gonzalez

# AURKIBIDEA:

# Kamera

- Fustrum Culling:

Zati hau egiteko bi funtzio osatu behar ditugu, Lehenengo funtzioa Scene/Node Klaseko frustumCull funtzioa izan da, bertan kamera bat izanda erabaki behar du uneko nodoa (eta bere umeak) frustum-aren barruan dagoen edo ez.Uneko nodoaren objektuaren m_isCulled eremua eguneratu behar du: eremu hori true bada, nodoa
(eta bere azpian dagoen azpi-zuhaitz osoa) frustum-etik kanpo dago, eta ez da errenderizatu behar. Eremua false bada, berriz, nodoa margotu behar da.

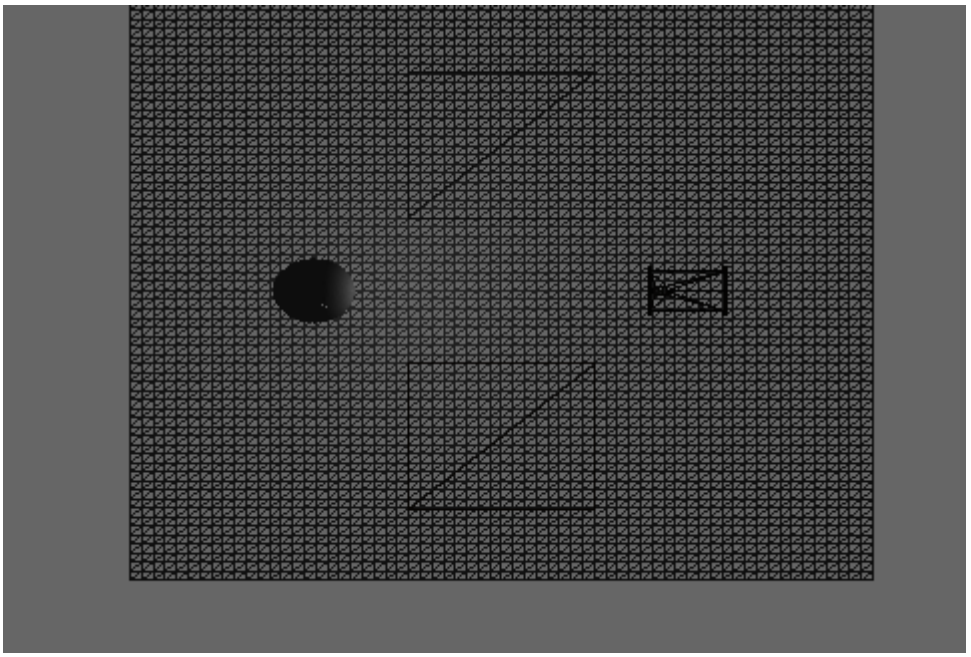
```
void Node::frustumCull(Camera *cam) {
    /* ================== PUT YOUR CODE HERE ====================== */
    unsigned int *planes;
    if (cam->checkFrustum(m_containerWC, 0)==1){    /*nodoa frustumaren kanpoan dagoen konprobatu*/
        m_isCulled=true;
    }else{  /*frustumaren kanpoan edo ebakitzen badu*/
        m_isCulled=false;
        this->draw();
    }
        for(auto & theChild : m_children) {
            theChild->frustumCull(cam);// Recursive call
        }
    /* ================== END YOUR CODE HERE ====================== */
}
```

Fustrum-aren barruan dagoela ziurtatzeko Kameran hurrengo funtzioa definitu dugu.

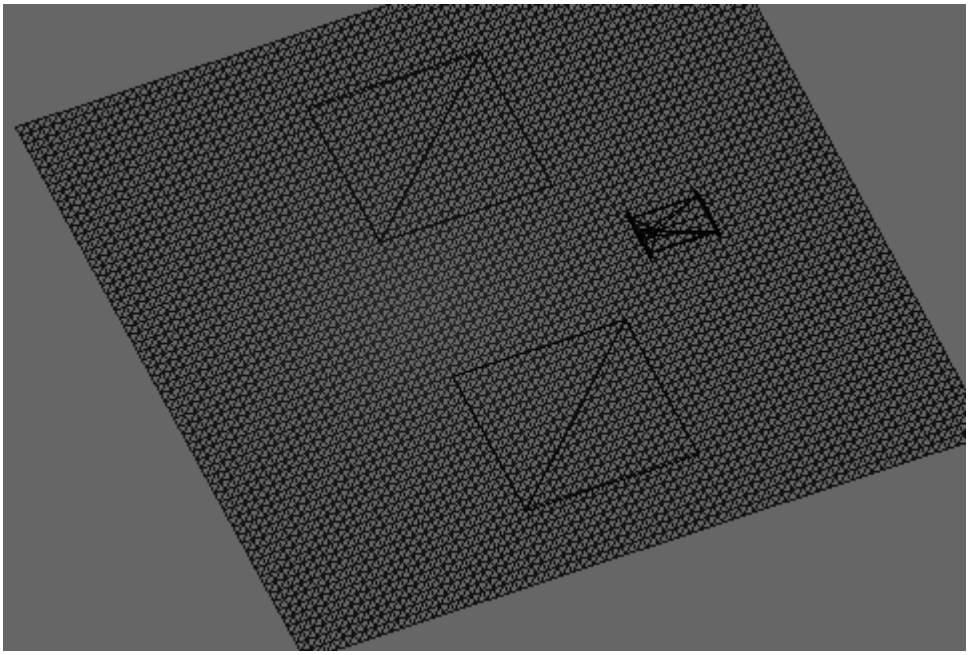
```
int Camera::checkFrustum(const BBox *theBBox,
                         unsigned int *planesBitM) {
    int res = -1; // by default, BBOX fully inside
    /* ==================== PUT YOUR CODE HERE ======================= */
    for(int i=0;i<MAX_CLIP_PLANES;i++) {
        res = BBoxPlaneIntersect(theBBox,m_fPlanes[i]);
        if(res==1){
            return 1;
        }
    }
    /* ==================== END YOUR CODE HERE ======================= */
    return res; // BBox is fully inside the frustum
}
```

Bertan munduko plano bakoitzeko eta objetuaren BoundingBox-a ebakitzen diren aztertuko dugu eta BBOx-a frustum-aren kanpoan badago, return 1 egingo dugu, hau da, ez da errenderizatuko.

Lortutako emaitza hurrengoa da:



Hau izango litzateke hasierako egoera, eta fustrum-a aldatu ostean:



# Argistatzea:

Zati honen helburua da eszena barruko argiak eta materialak kontuan hartzea. Horretarako, haibat aldaketa egin dizkiogu pervertex.vert eta .frag fitxategiei.

Lehenik eta behin 3 argi mota sortu ditugu, infinitua edo directionala, spotlight eta lokala, hasi baino lehen argien intentzitatea kalkulatzeko funtzioak sortu ditugu hurrengo formulak erabiliz:
Garrantzizua iruditzen zait formula hauek adieraztea lehenengo, argi guztietarako komunak direlako.

$$\mathbf{i}_{\text{diff}} = \mathbf{m}_{\text{diff}} \otimes \mathbf{s}_{\text{diff}}$$
$$\mathbf{i}_{\text{spec}} = \max(0, (\mathbf{r} \cdot \mathbf{v}))^m \mathbf{m}_{\text{spec}} \otimes \mathbf{s}_{\text{spec}}$$

- Directionala edo infinitua:

```
vec3 directional (int i){
    vec3 itot=vec3(0,0,0);
    vec4 n_4 =  modelToCameraMatrix * vec4(v_normal, 0);

    vec3 n = normalize(n_4).xyz; /*planoaren normala*/

    vec3 l = normalize(-theLights[i].position).xyz; /*argiaren kontrako norabidea,

    float angle = max(0,dot(l,n));  /*argia eta normalaren arteko angelua*/

    //barr
    vec3 idiff = theMaterial.diffuse * theLights[i].diffuse;    /*materialeko fakt
    //espek
    vec3 v = normalize(-(modelToCameraMatrix * (vec4(v_position,1)))).xyz; /*erpin
    vec3 r = 2*dot(l,n)*n - l;
    float angle_spec = pow(max(0, dot(r,v)),theMaterial.shininess);
    vec3 ispec = angle_spec * (theMaterial.specular * theLights[i].specular);   /*

    itot=angle * (idiff+ispec);
    return itot;

}
```

Argi hau hurrengo formulak erabiliz osatu da:

$$\mathbf{i}_{\text{tot}}^{i} = \max(0, \mathbf{n} \cdot \mathbf{l}_i) \cdot (\mathbf{i}_{\text{diff}} + \mathbf{i}_{\text{spec}})$$

non:

- $\mathbf{n}$: gainazalaren normala, normalizatua.
- $\mathbf{l}_i$: argiaren *kontrako* norabidea, normalizatua.
- $\max(0, \mathbf{n} \cdot \mathbf{l}_i) = \cos\theta$
    - argia eta normalaren arteko angelua.

- Argi lokala:

```
vec3 local (int i){
    vec3 itot=vec3(0,0,0);
    vec4 n_4 =  modelToCameraMatrix * vec4(v_normal, 0);

    vec3 n = normalize(n_4).xyz; /*planoaren normala*/

    vec3 l = normalize(-theLights[i].position).xyz; /*argiaren kontrako norabidea, normalizatua*/

    float angle = max(0,dot(l,n));  /*argia eta normalaren arteko angelua*/

    //barreiatua
    vec3 idiff = theMaterial.diffuse * theLights[i].diffuse;    /*materialeko faktore barreiatua bider argiren faktore barreiatua*/
    //espekularra
    vec3 v = normalize(-(modelToCameraMatrix * (vec4(v_position,1)))).xyz; /*erpinetik ikuslera doan bektore unitarioa*/
    vec3 r = 2*dot(l,n)*n - l;
    float angle_spec = pow(max(0, dot(r,v)),theMaterial.shininess);
    vec3 ispec = angle_spec * (theMaterial.specular * theLights[i].specular);   /*r eta v arteko angelua bider materialeko faktore espekularra

    vec3 P = (modelToCameraMatrix * (vec4(v_position,1))).xyz;  /*erpinaren posizioa*/
    vec3 Spos_p = (theLights[i].position.xyz-P);    /*argiaren posizioa - P*/
    vec3 li = normalize(Spos_p);
    float angle_local = max(0,dot(n, li));  /*gainazalaren normala eta rpinetik argira doan bektore unitarioaren arteko angelua*/
    float d = 1.0/(theLights[i].attenuation[0]+theLights[i].attenuation[1]*length(Spos_p) + theLights[i].attenuation[2]*pow(length(Spos_p),2));
    itot=d*angle_local*(idiff+ispec);
    return itot;
}
```

Hemen intentsitate totalaren formula aldatzen da,;

$$\mathbf{i}_{\text{tot}}^{i} = d \cdot \max(0, \mathbf{n} \cdot \mathbf{l}_i) \cdot (\mathbf{i}_{\text{diff}} + \mathbf{i}_{\text{spec}})$$

- $\mathbf{n}$ gainazalaren normala (normalizatua)
- $\mathbf{l}_i = \frac{\mathbf{s}_{\text{pos}} - \mathbf{p}}{\|\mathbf{s}_{\text{pos}} - \mathbf{p}\|}$ erpinetik argira doan bektore unitarioa
  - $\mathbf{p}$ erpinaren posizioa
  - $\mathbf{s}_{\text{pos}}$ argiaren posizioa
  - **Biak espazio berean**
- $d$: argiaren intentsitate-ahuldura, distantziarekiko proportzionala.

eta,

- Intentsitatearen ahuldura:

$$d = \frac{1}{s_c + s_l \| \mathbf{s}_{\text{pos}} - \mathbf{p} \| + s_q \| \mathbf{s}_{\text{pos}} - \mathbf{p} \|^2}$$

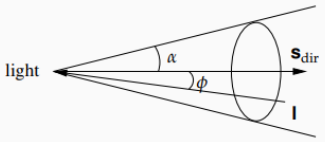
- Spotlight argia:

Egindako azkeneko argia da, aldaketa batzuekin argi hau puntu eta norabide batekaz definitzen dira, Lokala puntu batekaz bakarrik eta direkzionala norabidekaz.
Egindako kodea hurrengoa da:

```
vec3 spotlight (int i){
    vec3 itot=vec3(0,0,0);
    vec4 n_4 =  modelToCameraMatrix * vec4(v_normal, 0);
    vec3 n = normalize(n_4).xyz; /*planoaren normala*/
    vec3 l = normalize(-theLights[i].position).xyz; /*argiaren kontrako norabide
    float angle = max(0,dot(l,n));  /*argia eta normalaren arteko angelua*/
    //barreiatua
    vec3 idiff = theMaterial.diffuse * theLights[i].diffuse;    /*materialeko fa
    //espekularra
    vec3 v = normalize(-(modelToCameraMatrix * (vec4(v_position,1)))).xyz; /*erp
    vec3 r = 2*dot(l,n)*n - l;
    float angle_spec = pow(max(0, dot(r,v)),theMaterial.shininess);
    vec3 ispec = angle_spec * (theMaterial.specular * theLights[i].specular);

    vec3 P = (modelToCameraMatrix * (vec4(v_position,1))).xyz;  /*erpinaren posi
    vec3 Spos_p = (theLights[i].position.xyz-P);    /*argiaren posizioa - P*/
    vec3 li = normalize(Spos_p);
    float angle_local = max(0,dot(n, li));  /*gainazalaren normala eta rpinetik
    float angle_spot=dot(-li,theLights[i].spotDir);
    //spotlight
    float C_spot=0;
    if (angle_spot>theLights[i].cosCutOff){
        C_spot = max(angle_spot,0);
    }
    itot = pow(C_spot,theLights[i].exponent) * angle_local * (idiff+ispec);
    return itot;
}
```

Jakinik formulak hurrengoak direla,




- Kalkulatu $c_{\text{spot}} = \max(-\mathbf{l}_i \cdot \mathbf{s}_{\text{dir}}, 0) = \max(\cos\phi, 0)$
  - non $\mathbf{l}_i = \frac{\mathbf{s}_{\text{pos}} - \mathbf{p}}{\|\mathbf{s}_{\text{pos}} - \mathbf{p}\|}$
- $c_{\text{spot}}$ neurtzen du argi-norabide eta *spotlight* norabidearen arteko angeluaren kosinua. $c_{\text{spot}} < \cos\alpha$ bada, objektua *spotlight* konotik at dago, eta ez du intentsitaterik jasoko.
- Bestela (hau da, $c_{\text{spot}} > \cos\alpha$ bada):

$$\mathbf{i}_{\text{tot}}^{j} = c_{\text{spot}}^{\mathbf{s}_{\text{exp}}} \cdot \max(0, \mathbf{n} \cdot \mathbf{l}_i) \cdot (\mathbf{i}_{\text{diff}} + \mathbf{i}_{\text{spec}})$$

Argiztapenean lortutako emaitzak shaderren azalpenaren ostean aurkituko dira argazkiak hobeagoak izan daitezen.

Proiektuan argiak kudeatzeko eta haibat argi batera egoteko hurrengo main funtzioa sortu da:

```
    return itot;
}

void main() {
    vec3 itot = scene_ambient;
    for (int i=0; i<active_lights_n; i++){  /*piztutako argi kopurua
        if (theLights[i].position[3]==0){   /*puntu bat izan beharre
            itot += directional(i);
        }else if(theLights[i].cosCutOff>0 && theLights[i].cosCutOff<
            itot += spotlight(i);
        }else{
            itot += local(i);
        }
    }
    f_color=vec4(itot,1);
    f_texCoord = v_texCoord;
    gl_Position = modelToClipMatrix * vec4(v_position, 1);
}
```

Azkenik argiaz eszenan kokatu behar dira, horretarako funztio hau sortu dugu:

```
void Light::placeScene(const Trfm3D & view, const Trfm3D & model) {

    if( ! m_switched ) return;
    Trfm3D modelView = view * model; // this is the current modelview matrix

    /* ================== PUT YOUR CODE HERE ====================== */
        if (m_type==directional){
        m_positionEye = modelView.transformVector(m_position);
        m_positionEye.normalize();
    }
    if (m_type==positional){
        m_positionEye = modelView.transformPoint(m_position);
    }
    if (m_type==spotlight){
        m_positionEye = modelView.transformPoint(m_position);
        m_spotDirectionEye = modelView.transformVector(m_spotDirection);
        m_spotDirectionEye.normalize();
    }
    /* ================== END YOUR CODE HERE ====================== */
}
```

Hemen argiaren posizioa (edo norabidea) espazio lokaletik kameraren espaziora bihurtzen dituen. Halaber, spotlight motako argietan funtzioak spot-aren norabidea ere bihurtu behar du. Bihurketa egiteko, m position (eta m spotDirection) modelview matrizeekin biderkatu behar da, eta emaitza m positionEye (eta m spotDirectionEye) eremuan utzi.

## Shaderrak

Hemengo atalan aldaketa garrantsitzuenak jasan duten fitxategia pervertex.frag izan da, non varying diren bi atributu definiztu ditugu hasiera batean, bata texturaren koordenatuak daramana eta bestea kolorea objetuaren erpiinari ze kolore emango zaion.

```
varying vec4 f_color;
varying vec2 f_texCoord;
```

Koloreen mapeaketa burutzeko .frag fitxategiaren main-ean hurrengo kodea sartu da.

```
void main() {
    //gl_FragColor = vec4(1.0);
    vec4 texColor = texture2D(texture0, f_texCoord);
    // combine f_color and texColor
    gl_FragColor = texColor * f_color;

}
```

Azkenik errenderizatzerakoan mapeatutako textura aplikatzeko kode zatia sortu dugu:

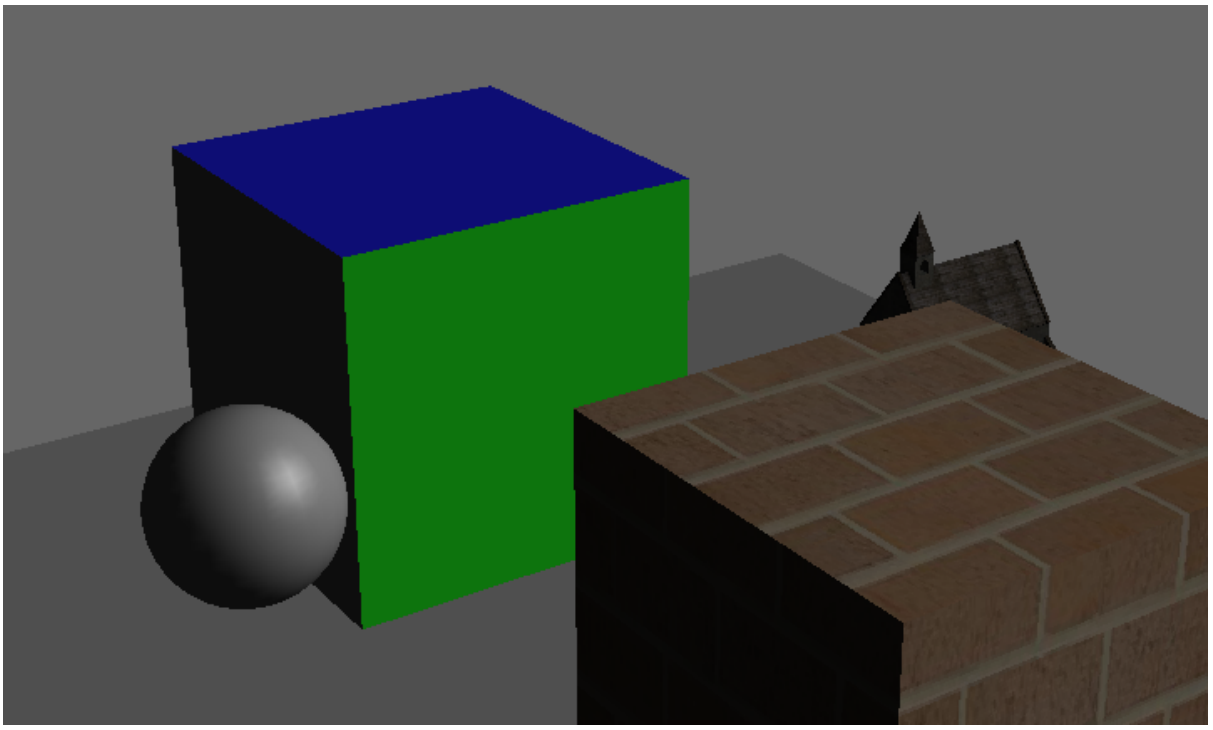
```
void ShaderProgram::beforeDraw() {

    Material *mat;
    Texture *tex;
```

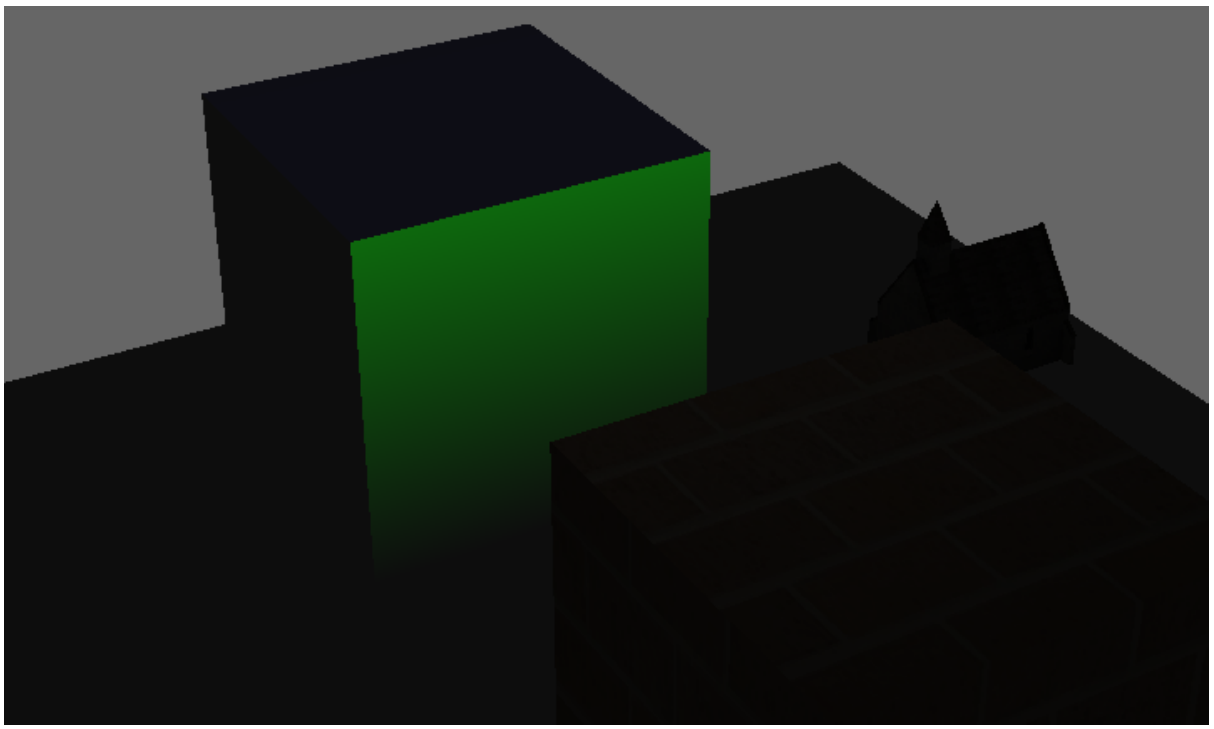
```
    tex = mat->getTexture();
    if (tex != 0) {
        // Set texture to unit 'Constants::gl_texunits::texture'
        tex->bindGLUnit(Constants::gl_texunits::texture);
        this->send_uniform("texture0", Constants::gl_texunits::texture);
    }
```

Orain hemen aurkeztuko dira shaderrak eta argien emaitzak:

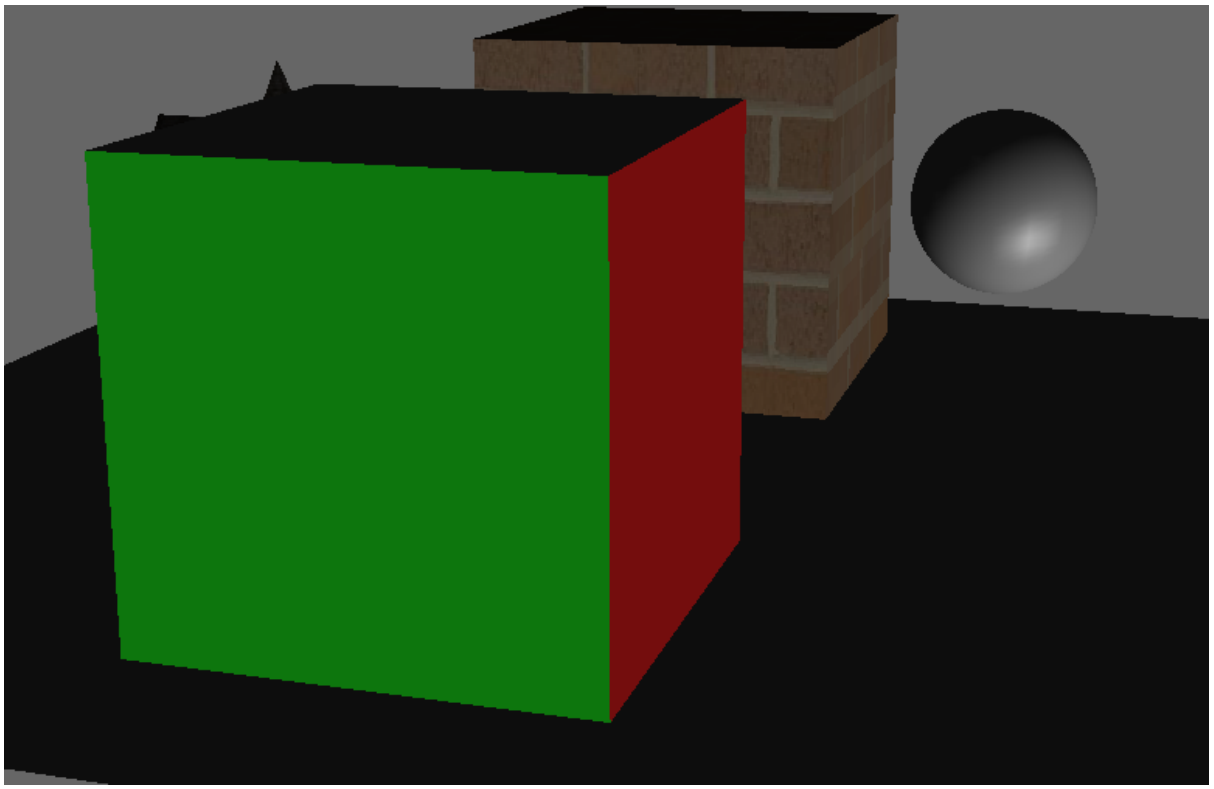
*Argi guztiak amatatuta ikusten dena inguruneko argia da.*



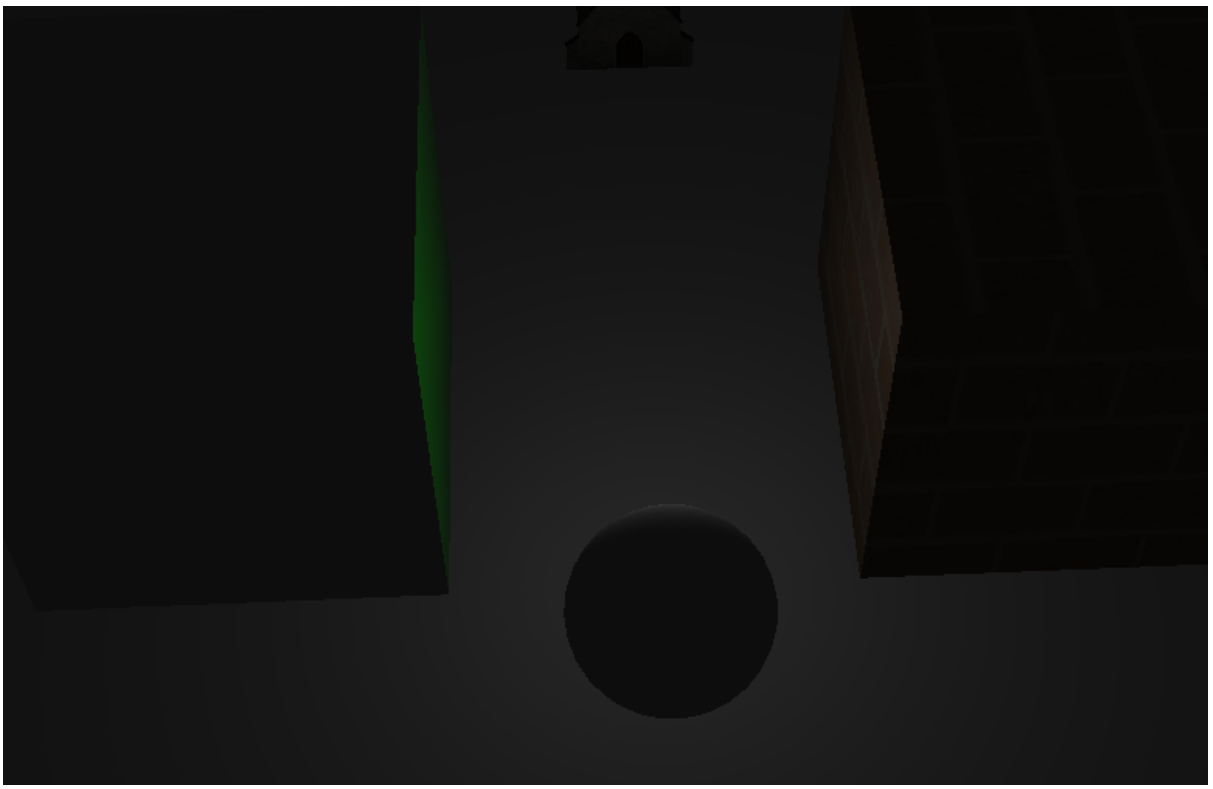
*Argi infinitua piztuta*



*Spotlight argia piztuta*



*2. argi direkzionala*



*Argi nagusia piztuta*

# Testura anitzak:

Hemengo zatian objetu bati bi textura aplikatuko dizkiogu, gure kasuan, esferari munfuko textura bat aplikatuko diogu eta gainean hodei batzuen textura aplikatuko diogu eta gainera hodei horiek bueltak emateko animazioa sortuko dugu.
Aurreko lana ez aldatzeko multitex.vert eta frag aldatuko ditugu.

Lehenik eta behin, Shader.cc/before draw() funtzioan, shader -ak multitex gaitasuna badu, esleitu bigarren testura Constants::gl texunits::rest unitatean. Material baten bigarren testura eskuratzeko, orduan hau gehitu behar dugu:

```
}
if (this->has_capability("multitex")) {
    Texture *tex2 = mat->getTexture(1);
```

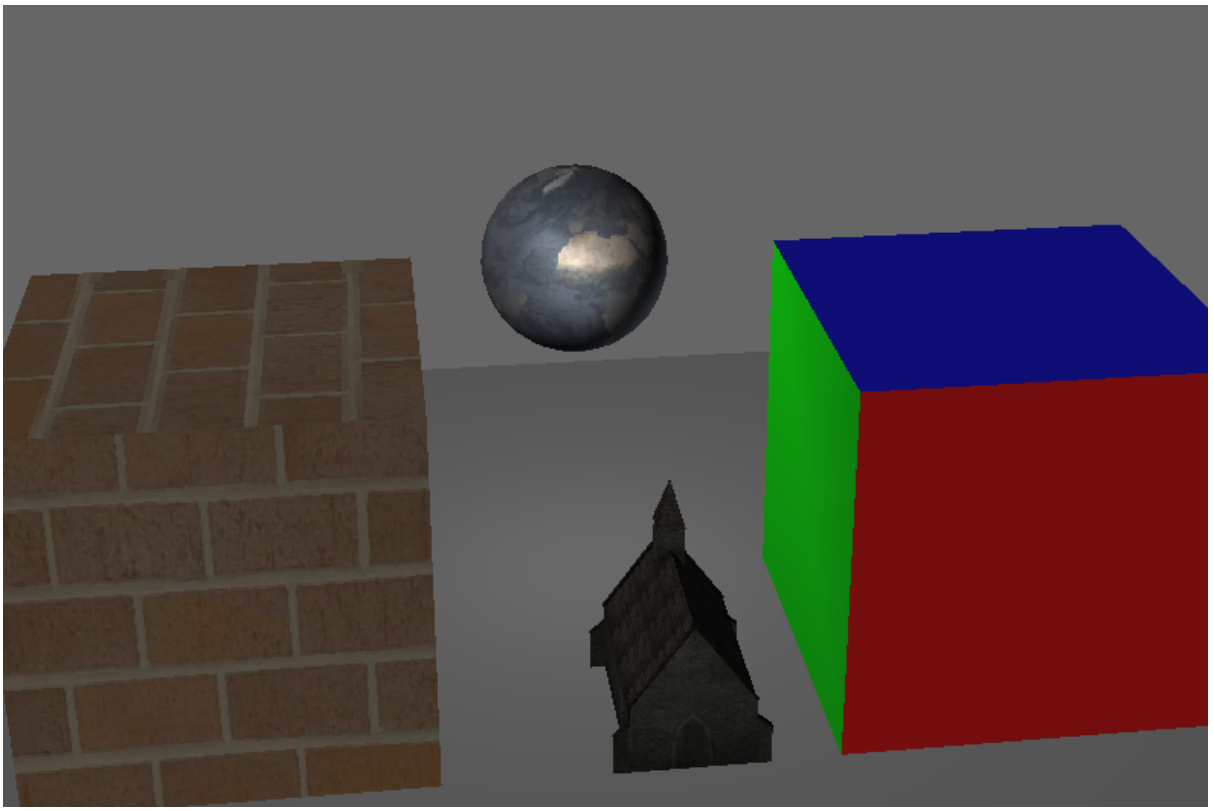
Ondoren pervertex multitex.frag shader -ean bi testurak hartu eta nahastu egin behar ditugu. Horretarako, bi testurak gehitu 0.5 faktore batekin, hau da,

```
uniform sampler2D texture0;
uniform sampler2D texture1;
```

```
texColor_0 = texture2D(texture0, f_texCoord);
texColor_1 = texture2D(texture1, f_texCoord_1);

// The final color must be a linear combination of both
// textures with a factor of 0.5, e.g:
//
//color = 0.5 * color_of_texture0 + 0.5 * color_of_texture1;
texColor_Tot = 0.5*texColor_0 + 0.5*texColor_1;
```

Lortutako emaitza hurrengoa da.



Amaitzeko 0 tekla sakatzerakoan esferan dauden lainoak mugitutzeko animazioa egingo dugu:
Lehengo Scene/RenderState objektuak atributu berri bat izan behar du m cloudsOffset, eta atributu hori aldatzeko set/get funtzio pare bat.

```
256        //m_cloudsOffset;
257        float m_cloudsOffset;
258    };
```

gero mugitzeko sc atributua ere gehituko dugu:

```
private:
    float m_sc;
```

Getterrak eta setterrak ere sortuko ditugu:

```
////////////////////////////////////////////
//          Get set m_cloudsOffset
void setm_cloudsOffset(float val);
float getm_cloudsOffset() const;

float getSc() const;
void setSc(float v);
```

Animazioa burutzeko sc aldagaia aldatu behar da ziklikoki, horretarako animate funtzioan m_cloudsOffset aldatu behar da.

```
RenderState::instance()->setm_cloudsOffset(p_cloudsOffset);
p_cloudsOffset+=0.0025;
if (p_cloudsOffset > 2){    /*cloudsOffseten balioa aldatu*/
    p_cloudsOffset=0;
}
```

Shader/before draw() funtzioan, shader -ak multitex gaitasuna badu, m cloudsOffset balioa pasa behar dio shaderari.

```
if ( class Texture ility("multitex")) {
    Texture *tex2 = mat->getTexture(1);
    if (tex2 != 0) {
        // bumpMapping in texture unit 1
        tex2->bindGLUnit(Constants::gl_texunits::rest);
        this->send_uniform("texture1", Constants::gl_texunits::rest);
        this->send_uniform("uCloudOffset",rs->getm_cloudsOffset()); /* m_cloudsOffset balioa pasa shader ari*/
    }
}
```

Zati honi amaiera emateko, Shaders/pervertex multitex.frag shader -ak bigarren testurari dagozkion koordenatuak mugituko ditu, S norabidean, uCloudOffset aldagaiaren arabera.

```
//gl_FragColor = vec4(1.0);
vec2 f_texCoord_1 = vec2(f_texCoord[0]+uCloudOffset,f_texCoord[1]); /
```