Iker Landarech - 241617
Marina Hernandez - 241273

Universitat Pompeu Fabra Barcelona

Lab 4 - VG
Advanced Computer Graphics

In this lab, we have been improving the volumetric data render and learning to control some aspects of the render process to improve the visualization.

The first aspect we have been working on is implementing both jittering approaches to our shader when computing the random offset to add to our first sample of each ray along its direction.

The first approach consisted of using a noise texture like *Blue_Noise* to obtain this random offset, meaning each of the rays' first samples will be reading different pixels from the noise texture. This first implementation can be observed in **Image 1** in the appendix.

Jittering second approach consisted of using a pseudorandom-looking function that uses the theory of waves and fractions to compute this offset to add to the first samples of our rays. This second approach implementation can be observed in **Image 2** in the appendix.

Results can be observed at **Jittering Outputs** at the end of this document.

The second aspect that we have worked on is implementing the transfer function. For that first, we created 6 TF_Textures.png each color design inspired by the volumes with size *[256x1]*. The first step consisted of applying the different TFs from the textures into the shader, and then the next step consisted of being able to use ImGui to control the change between the TFs with a Combo selector. The implementation of this selector was a bit expensive since we needed to check 36 different cases between models and TFs so we have been working on refactoring the implementation simplifying the logic. This implementation can be observed in **Image 3** in the appendix.

Before continuing with the next step in the lab, as can be seen in **Image 4** in the appendix we have been working on some ImGui preset buttons for each of the volumes so results can be easily observed at runtime.

For the third aspect, we have worked on implementing the volume clipping, meaning modifying the fragment shader so that if the sample point is on the side we want to hide it is not shown. For that, we have also implemented some ImGui sliders to control all directions from the clipping plane into the *VolumeNode*. This implementation can be observed in **Image 5** in the appendix.

The last challenge of this lab was to work on implementing isosurface visualizations. For that, the first thing we did was implement a gradient function that returns the normal perpendicular to the surface by computing the partial derivatives and a constant “h”.The implementation of the gradient function can be observed in **Image 6** in the appendix. Logically we have passed the corresponding uniforms and created the corresponding definitions into the shader to make it work. As well as the ImGui slider implementations for controlling the *gradient threshold* and the *h* constant value.

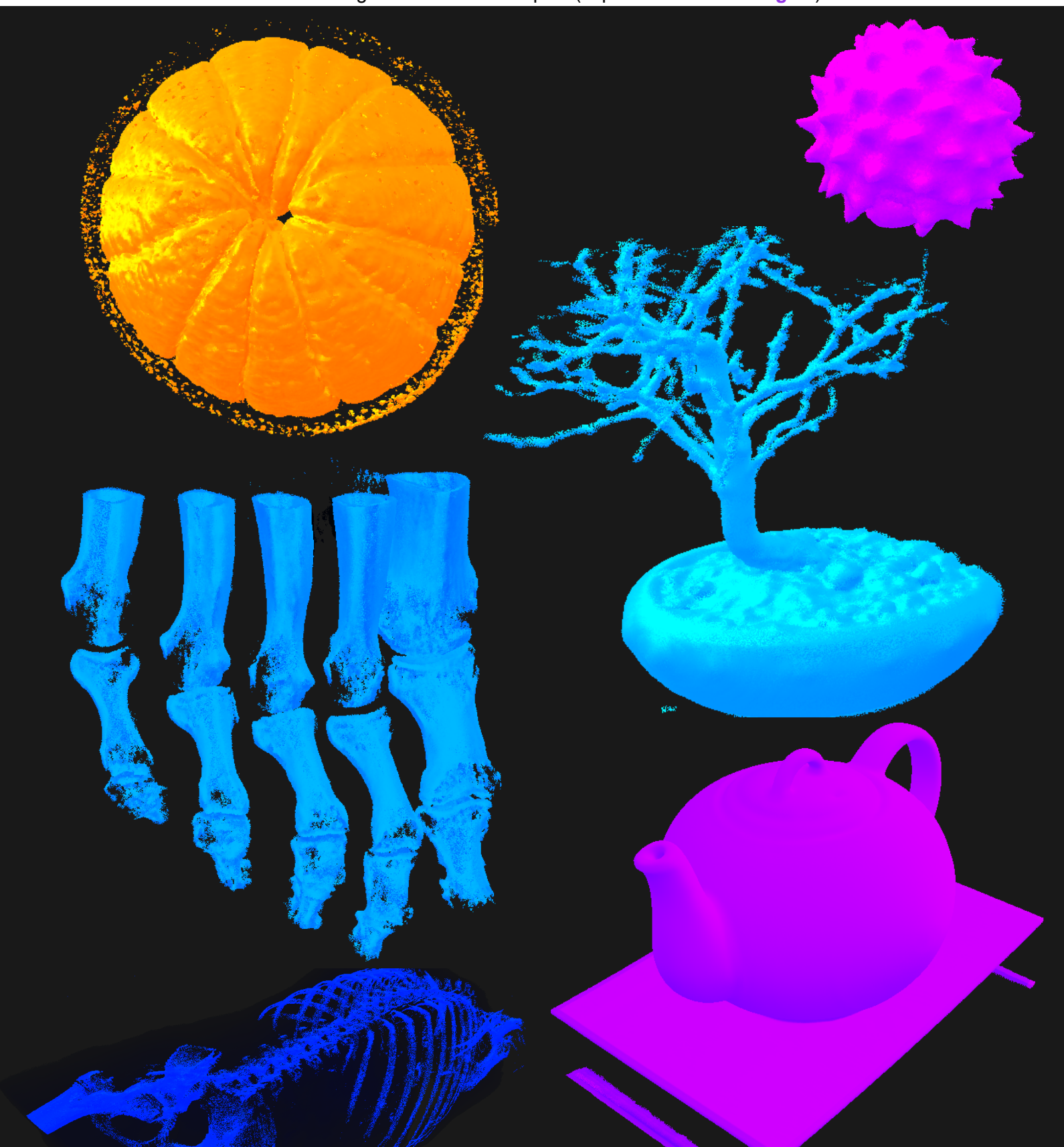
Iker Landarech - 241617
Marina Hernandez - 241273

upf. Universitat Pompeu Fabra Barcelona

The next step for isosurface visualizations was to implement a function for computing the simple shading equation in order to apply *Phong*. This implementation can be observed in **Image 7** in the appendix.

After computing both gradient and phong color methods we were able to finally compute the final color whose implementation can be observed in **Image 8** in the appendix.

Finally, we implemented 6 more presets for the isosurfaces which set all the settings we need to obtain the following visualization examples (implementation at **Image 9**):



Iker Landarech - 241617
Marina Hernandez - 241273

upf. Universitat Pompeu Fabra Barcelona

APPENDIX:

Image 1: jittering_1

C/C++

material.cpp:

```
    noise_texture = Texture::Get("data/images/blueNoise.png");
    shader->setUniform("noise_texture", noise_texture);
```

volumeshader.fs:

```
    uniform sampler2D noise_texture; //noise texture
    uniform bool jittering_1; //blue noise texture approach

    vec3 offset_1 = texture2D(noise_texture, gl_FragCoord.xy/128.0).xyz;
    if(jittering_1){ //blue noise texture approach
       sample_pos += offset_1 * step_vector;
    }
```

Image 2: jittering_2

C/C++

material.cpp:

```
    noise_texture = Texture::Get("data/images/blueNoise.png");
    shader->setUniform("noise_texture", noise_texture);
```

volumeshader.fs:

```
    uniform bool jittering_2; //pseudorandom-looking function approach

    float rand(vec2 co){ //jittering pseudorandom-looking function.
       return fract(sin(dot(co,vec2(12.9898, 78.233)))*43758.5453);
    }
    vec2 co =gl_FragCoord.xy; //rand input
    float offset_2 = rand(co); //function defined above

    if(jittering_2){ //pseudorandom-looking function approach
       sample_pos += offset_2 * step_vector;
    }
```

Image 3: Transfer Function

C/C++

material.h:

```
        Texture* TF_abdomen = NULL;
        Texture* TF_bonsai = NULL;
        Texture* TF_foot = NULL;
        Texture* TF_daisy = NULL;
        Texture* TF_orange = NULL;
        Texture* TF_teapot = NULL;
```

material.cpp:

```
VolumeMaterial::VolumeMaterial()
{
        color = vec4(1.f, 1.f, 1.f, 1.f);
        shader = Shader::Get("data/shaders/basic.vs", "data/shaders/volumeshader.fs");
        noise_texture = Texture::Get("data/images/blueNoise.png", true, GL_CLAMP_TO_EDGE);
        TF_texture = Texture::Get("data/images/TF_abdomen.png", true, GL_CLAMP_TO_EDGE);
        TF_abdomen = Texture::Get("data/images/TF_abdomen.png", true, GL_CLAMP_TO_EDGE);
        TF_foot = Texture::Get("data/images/TF_foot.png", true, GL_CLAMP_TO_EDGE);
        TF_orange = Texture::Get("data/images/TF_orange.png", true, GL_CLAMP_TO_EDGE);
        TF_daisy = Texture::Get("data/images/TF_daisy.png", true, GL_CLAMP_TO_EDGE);
        TF_bonsai = Texture::Get("data/images/TF_bonsai.png", true, GL_CLAMP_TO_EDGE);
        TF_teapot = Texture::Get("data/images/TF_teapot.png", true, GL_CLAMP_TO_EDGE);
}

void VolumeMaterial::renderInMenu()
{
   unsigned int text_current = 0;


    Texture* textures[] = {TF_abdomen, TF_daisy, TF_orange, TF_bonsai,    TF_foot,
TF_teapot};
    const char* textures_def[] = {"TF_Abdomen", "TF_Daisy", "TF_Orange",
"TF_Bonsai", "TF_Foot", "TF_Teapot"};

if (item_current >= 0 && item_current < 6) {
        Texture* selectedTexture = textures[text_current >= 0 && text_current < 6 ?
text_current : 0];

        //ImGui Change between textures:
material.cpp:
        switch (item_current) {
        case 0: this->texture = texture_abdomen; break;
        case 1: this->texture = texture_daisy; break;
        case 2: this->texture = texture_orange; break;
        case 3: this->texture = texture_bonsai; break;
        case 4: this->texture = texture_foot; break;
        case 5: this->texture = texture_teapot; break;
        }

        this->TF_texture = selectedTexture;
        }
ImGui::Checkbox("Transfer Function", &transfer);
```

```
        ImGui::Combo("Choose the Transfer Function", (int*)&text_current,
textures_def, IM_ARRAYSIZE(textures_def));
```

volumeshader.fs:

```
    uniform sampler2D TF; // transfer function texture

    vec4 colorTF = texture2D(TF,vec2(d,1)); //vec2(d,0.5)
    sample_color *= colorTF;
```

Image 4: ImGui Preset Buttons

C/C++

```
        if (ImGui::Button("Abdomen Preset")) {
                abdomenPreset();
        }
        if (ImGui::Button("Daisy Preset")) {
                daisyPreset();
        }
        if (ImGui::Button("Orange Preset")) {
                orangePreset();
        }
        if (ImGui::Button("Bonsai Preset")) {
                bonsaiPreset();
        }
        if (ImGui::Button("Foot Preset")) {
                footPreset();
        }
        if (ImGui::Button("Teapot Preset")) {
                teapotPreset();
        }
}

//PRESETS:
void VolumeMaterial::abdomenPreset()
{
        u_color = vec4(1.0, 1.0, 1.0, 1.0);
        isosurface = false;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.02;
        brightness = 6.2;
        alpha_discard = 0.085;
```

```
        text_current = 0;
        item_current = 0;
}

void VolumeMaterial::daisyPreset()
{
        u_color = vec4(1.0, 1.0, 1.0, 1.0);
        isosurface = false;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.02;
        brightness = 6.2;
        alpha_discard = 0.1;
        text_current = 1;
        item_current = 1;
}

void VolumeMaterial::orangePreset()
{
        u_color = vec4(1.0, 1.0, 1.0, 1.0);
        isosurface = false;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.02;
        brightness = 6.2;
        alpha_discard = 0.123;
        text_current = 2;
        item_current = 2;
}

void VolumeMaterial::bonsaiPreset()
{
        u_color = vec4(1.0, 1.0, 1.0, 1.0);
        isosurface = false;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.0373;
        brightness = 7.5;
```

```
        alpha_discard = 0.085;
        text_current = 3;
        item_current = 3;
}

void VolumeMaterial::footPreset()
{
        u_color = vec4(1.0, 1.0, 1.0, 1.0);
        isosurface = false;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.02;
        brightness = 6.2;
        alpha_discard = 0.09;
        text_current = 4;
        item_current = 4;
}

void VolumeMaterial::teapotPreset()
{
        u_color = vec4(1.0, 1.0, 1.0, 1.0);
        isosurface = false;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.035;
        brightness = 10;
        alpha_discard = 0.035;
        text_current = 5;
        item_current = 5;
}
```

Image 5: Volume Clipping

C/C++

material.cpp:

```
    shader->setUniform("clipping", clipping);
    shader->setUniform("plane_clipping", plane_clipping);

    ImGui::Checkbox("Clipping", &clipping);
    ImGui::SliderFloat3("Slider Clipping X-Y-Z", &plane_clipping.x, 0, 1);
    ImGui::SliderFloat("Slider Clipping W", &plane_clipping.w, -1, 1);
```

volumeshader.fs:

```
     if(clipping){
           float clip = sample_pos.x * plane_clipping.x + sample_pos.y *
plane_clipping.y + sample_pos.z * plane_clipping.z + plane_clipping.w;
               vec4 aux_color = final_color;
           if(clip > 0.0){ //Checking if the sample point is in the side we want to hide.
               final_color = aux_color;
                       }
     else {
        final_color += u_step * (1.0 - final_color.a) * sample_color;
```

Image 6: Gradient Implementation

C/C++

volumeshader.fs:

```
//Gradient Function:
vec3 gradient(vec3 s_position_tex) {

    vec3 gradient = vec3(0.0, 0.0, 0.0);

    //Computing the partial derivatives in x,y,z directions:
    float dx0 = texture(u_texture, s_position_tex + vec3(h, 0.0, 0.0)).x;
    float dx1 = texture(u_texture, s_position_tex + vec3(-h, 0.0, 0.0)).x;
    float dy0 = texture(u_texture, s_position_tex + vec3(0.0, h, 0.0)).x;
    float dy1 = texture(u_texture, s_position_tex + vec3(0.0, -h, 0.0)).x;
    float dz0 = texture(u_texture, s_position_tex + vec3(0.0, 0.0, h)).x;
    float dz1 = texture(u_texture, s_position_tex + vec3(0.0, 0.0, -h)).x;

    //Computing the gradient components using central differences:
    gradient.x = (dx0 - dx1) / (2.0 * h);
    gradient.y = (dy0 - dy1) / (2.0 * h);
    gradient.z = (dz0 - dz1) / (2.0 * h);

    //Normalizing the gradient vector so that it is a unit vector:
    gradient = normalize(gradient);

    //Inverting the gradient to obtain the surface normal:
    vec3 normal = 1.0 - gradient;

    return normal;
}
```

Image 7: Phong Implementation

```C/C++
//Computing the Phong lighting model for a given normal vector and texture position:
vec3 phong(vec3 v_normal, vec3 s_position_tex) {
  //Reading the pixel RGBA from the texture at position v_uv:
  vec4 color = vec4(u_color, 1.0);

  //Normalizing the surface normal vector.:
  vec3 N = normalize(v_normal);

  //Computing the normalized direction vector from the light source to the surface:
  vec3 L = normalize(light_position - s_position_tex);

  //Computing the dot product of the normalized light direction and the surface
normal:
  float LN = dot(L, N);
  LN = clamp(LN, 0.0, 1.0);

  //Computing the normalized view direction vector from the surface to the camera:
  vec3 V = normalize(u_camera_position - s_position_tex);

  //Computing the reflected light direction:
  vec3 R = normalize(reflect(-L, N));

  //Computing the dot product of the reflected light direction and the view
direction:
  float RV = dot(R, V);
  RV = clamp(RV, 0.0, 1.0);

  //Computing the specular reflection coefficient based on the texture color alpha
channel:
  vec3 ks = color.xyz * color.w;

  //Computing the ambient, diffuse, and specular components of the Phong model:
  vec3 ambient = ka * ambient_light * color.xyz;
  vec3 diffuse = kd * (LN) * diffuse_light * color.xyz;
  vec3 specular = ks * pow(RV, alpha) * specular_light;

  //Computing the final Phong lighting intensity:
  vec3 Ip = ambient + diffuse + specular;

  return Ip;
}
```

Image 8: Isosurface (Shader):

```C/C++
   if (isosurface){ //Isosurface [activate from ImGui]
              if (d > gradient_threshold){
                     normal = gradient(s_position_text);
                     phong_color = vec4(phong(normal,s_position_text), 1.0);
       if (clipping){
       float clip = sample_pos.x * plane_clipping.x + sample_pos.y *
plane_clipping.y + sample_pos.z * plane_clipping.z + plane_clipping.w
                     vec4 aux_color = final_color;
       if (clip > 0.0){ //Checking if the sample point is in the side we want to hide.
              final_color = aux_color;
       }
       else {
              final_color = phong_color;
       }
       break;
       }
   }
}
```

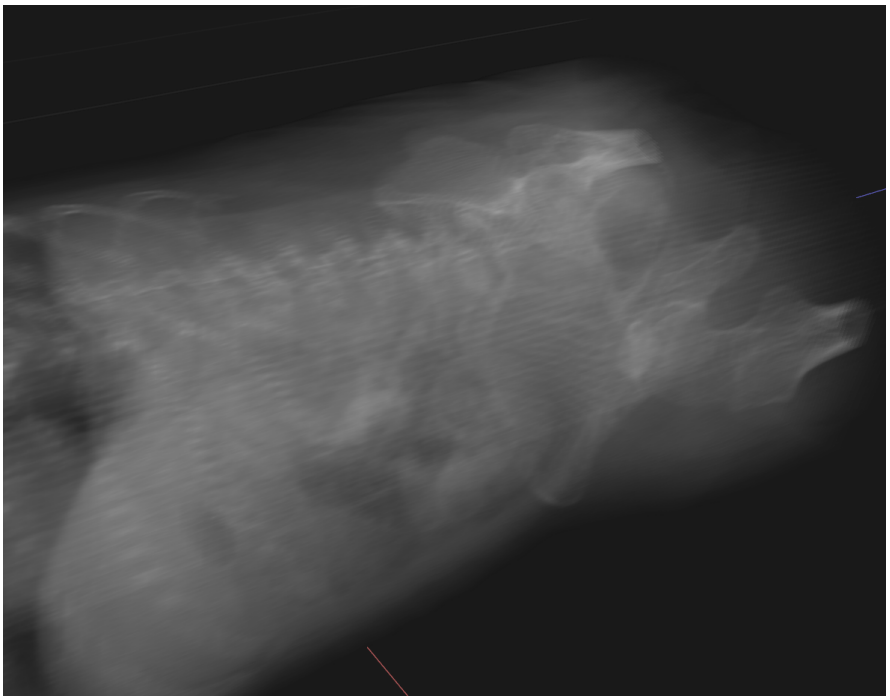
Image 9: Isosurface Presets:

```C/C++
//ISOSURFACES PRESETS:
void VolumeMaterial::abdomenIsoPreset()
{
       plane_clipping.x = 0.0;
       plane_clipping.y = 0.0;
       plane_clipping.z = 1.0;
       plane_clipping.w = -1.0;
       isosurface = true;
       u_color = vec4(0.0, 0.059, 1.0, 1.0);
       gradient_threshold = 0.402;
       jittering_2 = true;
       transfer = false;
       clipping = true;
       step = 0.02;
       brightness = 1.73;
       alpha_discard = 0.150;
       text_current = 0;
       item_current = 0;
       float h = 0.015;
}

void VolumeMaterial::daisyIsoPreset()
{
       isosurface = true;
       plane_clipping.x = 0.0;
       plane_clipping.y = 0.0;
       plane_clipping.z = 1.0;
       plane_clipping.w = -1.0;
       u_color = vec4(0.088, 0.0, 1.0, 1.0);
```

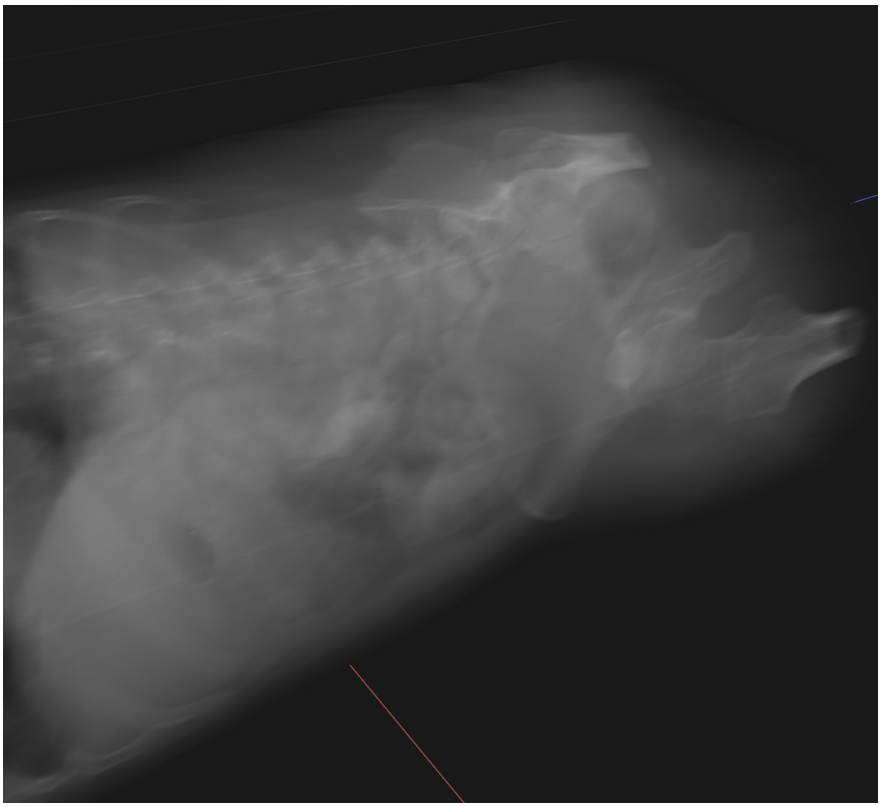
```
        gradient_threshold = 0.185;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.02;
        brightness = 6.2;
        alpha_discard = 0.1;
        text_current = 1;
        item_current = 1;
        float h = 0.015;
}

void VolumeMaterial::orangeIsoPreset()
{
        isosurface = true;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        u_color = vec4(1.0, 0.059, 0.0, 1.0);
        gradient_threshold = 0.211;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.02;
        brightness = 6.2;
        alpha_discard = 0.123;
        text_current = 2;
        item_current = 2;
        float h = 0.015;
}

void VolumeMaterial::bonsaiIsoPreset()
{
        isosurface = true;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        u_color = vec4(0.0, 0.059, 1.0, 1.0);
        gradient_threshold = 0.236;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.0373;
        brightness = 7.5;
        alpha_discard = 0.085;
        text_current = 3;
        item_current = 3;
        float h = 0.015;
}

void VolumeMaterial::footIsoPreset()
{
        isosurface = true;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        u_color = vec4(0.0, 0.059, 1.0, 1.0);
        gradient_threshold = 0.421;
```

```
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.02;
        brightness = 6.2;
        alpha_discard = 0.150;
        text_current = 4;
        item_current = 4;
        float h = 0.015;
}

void VolumeMaterial::teapotIsoPreset()
{
        isosurface = true;
        plane_clipping.x = 0.0;
        plane_clipping.y = 0.0;
        plane_clipping.z = 1.0;
        plane_clipping.w = -1.0;
        u_color = vec4(0.059, 0.0, 1.0, 1.0);
        gradient_threshold = 0.148;
        jittering_2 = true;
        transfer = true;
        clipping = true;
        step = 0.004;
        brightness = 7.3;
        alpha_discard = 0.05;
        text_current = 5;
        item_current = 5;
        float h = 0.015;
}
```

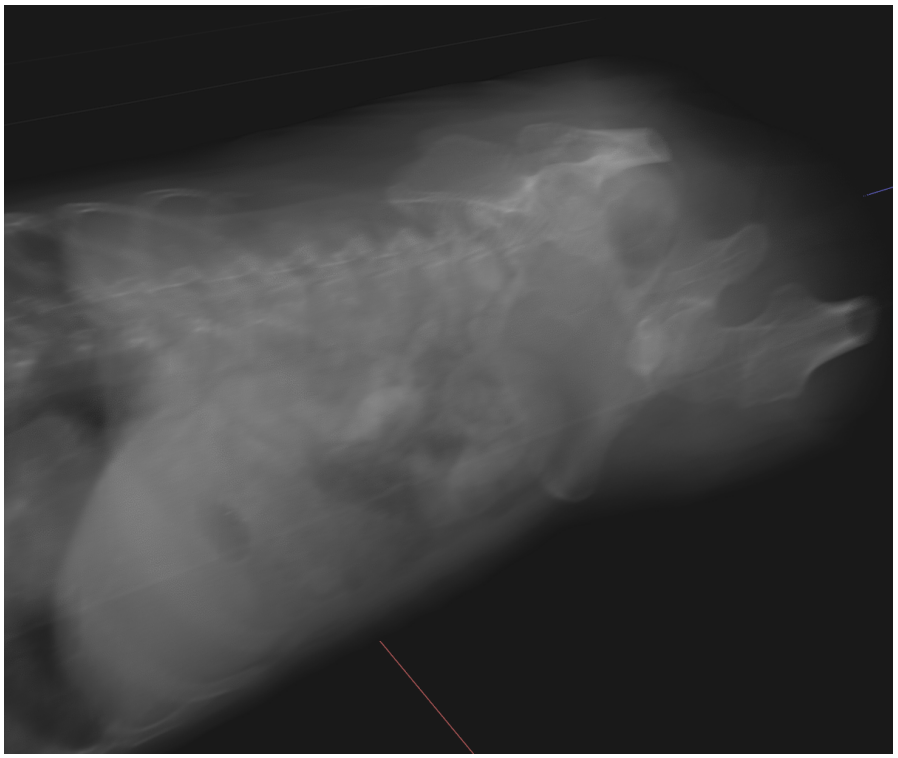
Jittering Outputs:



No Jittering



Blue Noise Texture-based Jittering



Pseudorandom-looking function-based Jittering