International Journal of Innovation Scientific Research and Review
Vol. 05, Issue, 11, pp.5493-5499, November 2023
Available online at http://www.journalijisr.com
SJIF Impact Factor 2023: 6.599

IJISRR

ISSN: 2582-6131

Research Article

COMPARISON MSAA ALGORTIHM AND FXAA ALGORITHM IN COMPUTER GRAPHICS

*** Charisma Tubagus Setyobudhi**

Department of Computer Science, Faculty of Technology and Engineering, Universitas Dipoengeoro, JL. Prof. H. Soedarto, SH Tembalang Semarang Indonesia.

Received 20th September 2023; Accepted 21th October 2023; Published online 30th November 2023

ABSTRACT

Anti Aliasing is a technique in which a pixelated line is changed to be smoother in computer graphics. Anti Aliasing is needed in computer graphics so that images look more realistic. There are several types of anti-aliasing techniques, namely, MSAA, FXAA, TAA, and SSAA. In this paper, the author compares the performance of two algorithms, namely MSAA and FXAA, using Open GL as their implementation. The comparison method used to compare the two algorithms is the visual quality and performance produced by the two algorithms.

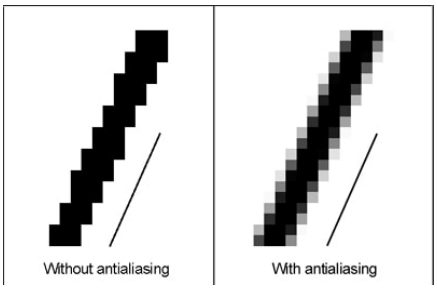
Keywords: Anti aliasing, MSAA, FXAA.

BACKGROUND

Anti Aliasing Anti-aliasing has been a subject of research in the computer field for more than 40 years [1]. Anti-aliasing is used in computer graphics to prevent or reduce the effects of aliasing. The aliasing effect occurs when an uneven edge (ladder-shaped) appears in the image. This is because the rasterisation process is distorted by sampling at a very low frequency. Under sampling is also called under sampling. Under sampling occurs because the sampling is performed at a frequency lower than the Nyquist frequency. The minimum sampling frequency (fs) so that under sampling does not occur can be formulated as follows:

$$Fs = 2 * F_{max}$$

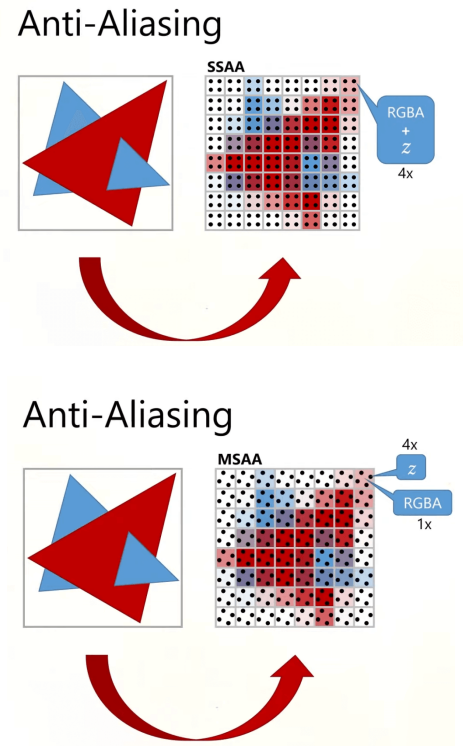
In real terms, we can see aliasing and anti-aliasing effects in the image below.



Several techniques or algorithms can be used for anti-aliasing, including MSAA, FXAA, SMAA, TAA, MLAA[2], RSAA[3], and NSAA[4]. Anti-aliasing implementation is usually found in graphics hardware or GPU [5].

MSAA (Multi Sampling Anti Aliasing)

In MSA, super sampling is performed. Super sampling is performed by rendering the scene at a higher resolution and down sampling it to an output with a lower resolution. In general, all Graphical Processing Units (GPUs) have similar mechanisms. MSAA can be performed with several samples, namely 2x, 4x, 8x, 16x. The MSAA technique is similar to or similar to super sampling anti-aliasing (SSAA), where SSAA also performs a similar process. However, there is a difference between MSAA and SSAA, namely in the shade section where MSAA executes the fragment shader only once per pixel. Meanwhile, the SSAA executes the fragment shader several times for each pixel. This causes the SSAA to behavior than the MSAA. Another difference is that the MSAA super samples only the edges, whereas the SSAA super samples the entire scene.



FXAA (Fast Approximate Anti Aliasing)

The FXAA is an AA algorithm created by NVIDIA. This algorithm is a post-processing technique in which the rendered frame is detected and which part of the screen needs to be smoothed. FXAA was performed to smooth the jaggy edges using a colour approach. The FXAA technique is faster than MSAA; however, the resulting image is

***Corresponding Author: Charisma Tubagus Setyobudhi,**
Department of Computer Science, Faculty of Technology and Engineering, Universitas Dipoengeoro, SH Tembalang Semarang Indonesia.

slightly blurry. The advantage of the FXAA technique is that the processing speed is relatively fast, button the other hand, the resulting image is slightly reduced in quality.

SMAA (Sub pixel Morphological Anti Aliasing)

SMAA is a technique that was developed base don MLAA.MLAA is a post-processing technique. SMAA uses a GPU instead of a CPU, unlike MLAA. The SMAA technique detects edges and uses filtering to smoothen them. The SMAA's performance is balanced enough for use, which is not too burdensome for the GPU, and the results produced are quite smooth.

TAA (Temporal Anti Aliasing)

The TAA uses time to help smooth the jagged edges. TAA looks at the previously rendered image in the buffer to determine which edges are smooth, rather than analysing the pixel so fan image.

```
voidWindow::Update()
    {
        for (inti = 0; i<Input::NUM_MOUSEBUTTONS; i++)
        {
                m_input.SetMouseDown(i, false);
                m_input.SetMouseUp(i, false);
        }
        for(inti = 0; i<Input::NUM_KEYS; i++)
        {
                m_input.SetKeyDown(i, false);
                m_input.SetKeyUp(i, false);
        }
        SDL_Evente;
        while(SDL_PollEvent(&e))
        {
                if(e.type == SDL_QUIT)
                {
                        m_isCloseRequested = true;
                }
                if(e.type == SDL_MOUSEMOTION)
                {
                        m_input.SetMouseX(e.motion.x);
                        m_input.SetMouseY(e.motion.y);
                }
                if(e.type == SDL_KEYDOWN)
                {
                        int value = e.key.keysym.scancode;

                        m_input.SetKey(value, true);
                        m_input.SetKeyDown(value, true);
                }
                if(e.type == SDL_KEYUP)
                {
                        int value = e.key.keysym.scancode;
                        m_input.SetKey(value, false);
                        m_input.SetKeyUp(value, true);

                }
```

RESEARCH METHODS

In this article, the author compares two methods or techniques for anti-aliasing, namely MSAA and FXAA. Open GL will be used as a tool or library for the implementation of these two techniques.

To implement anti-aliasing, the author created the components in the game engine that will be used. These components are as follows:

1. Window

The Window component creates a window object in which the scene will be drawn. This component has several important functions, namely Update() and Swap Buffers()

Function Update() inside the window

```
		if(e.type == SDL_MOUSEBUTTONDOWN)
			{
				int value = e.button.button;
				m_input.SetMouse(value, true);
				m_input.SetMouseDown(value, true);
			}
			if(e.type == SDL_MOUSEBUTTONUP)
			{
				int value = e.button.button;
				m_input.SetMouse(value, false);
				m_input.SetMouseUp(value, true);
			}
		}
	}
	voidWindow::Update()
	{
		for(inti = 0; i<Input::NUM_MOUSEBUTTONS; i++)
		{
			m_input.SetMouseDown(i, false);
			m_input.SetMouseUp(i, false);
		}
		for(inti = 0; i<Input::NUM_KEYS; i++)
		{
			m_input.SetKeyDown(i, false);
			m_input.SetKeyUp(i, false);
		}
		SDL_Evente;
		while(SDL_PollEvent(&e))
		{
			if(e.type == SDL_QUIT)
			{
				m_isCloseRequested = true;
			}
			if(e.type == SDL_MOUSEMOTION)
			{
				m_input.SetMouseX(e.motion.x);
				m_input.SetMouseY(e.motion.y);
			}
			if(e.type == SDL_KEYDOWN)
			{
				int value = e.key.keysym.scancode;
				m_input.SetKey(value, true);
				m_input.SetKeyDown(value, true);
			}
			if(e.type == SDL_KEYUP)
			{
				int value = e.key.keysym.scancode;
				m_input.SetKey(value, false);
				m_input.SetKeyUp(value, true);
			}
```

```
if(e.type == SDL_MOUSEBUTTONDOWN)
        {
                int value = e.button.button;

                m_input.SetMouse(value, true);
                m_input.SetMouseDown(value, true);
        }
        if(e.type == SDL_MOUSEBUTTONUP)
        {
                int value = e.button.button;
                m_input.SetMouse(value, false);
                m_input.SetMouseUp(value, true);
        }
  }
}
```

Fungsi SwapBuffers() di dalam Window

```
voidWindow::SwapBuffers()
{
  SDL_GL_SwapWindow(m_window);
}
```

2. Core Engine

The main task of the Core Engine component was to first connect the window to b ecreated and the game to be rendered. There are several functions of Core Engine:

FunctionCreateWindow()

```
voidCoreEngine::CreateWindow(const std::string&title)
{
  m_window = newWindow(m_width, m_height, title);
  m_renderingEngine = newRenderingEngine(*m_window);
}
```

Function Start()

```
voidCoreEngine::Start()
{
  if(m_isRunning)
  {
        return;
  }
  Run();
}
```

Function Stop()

```
voidCoreEngine::Stop()
{
  if(!m_isRunning)
  {
        return;
  }
  m_isRunning = false;
}
```

3. Rendering Engine

Rendering Engine is the component in which the main function for rendering is executed. Rendering Engine combines several important elements, namely, Game Object, Lighting, and Camera. There are severa limportant functions of the Rendering Engine:

```
voidRenderingEngine::Render(constGameObject&object, constCamera&mainCamera)
{
  GetTexture("displayTexture").BindAsRenderTarget();
  //m_window->BindAsRenderTarget();
  //m_tempTarget->BindAsRenderTarget();

  glClearColor(0.0f,0.0f,0.0f,0.0f);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  object.RenderAll(m_defaultShader, *this, mainCamera);

  for(unsignedinti = 0; i<m_lights.size(); i++)
  {
        m_activeLight = m_lights[i];
        ShadowInfoshadowInfo = m_activeLight->GetShadowInfo();

        intshadowMapIndex = 0;
        if(shadowInfo.GetShadowMapSizeAsPowerOf2() != 0)
              shadowMapIndex = shadowInfo.GetShadowMapSizeAsPowerOf2() - 1;

        assert(shadowMapIndex>= 0 &&shadowMapIndex< NUM_SHADOW_MAPS);

        SetTexture("shadowMap", m_shadowMaps[shadowMapIndex]);
        m_shadowMaps[shadowMapIndex].BindAsRenderTarget();
        glClearColor(1.0f,1.0f,0.0f,0.0f);
        glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

        if(shadowInfo.GetShadowMapSizeAsPowerOf2() != 0)
        {
              m_altCamera.SetProjection(shadowInfo.GetProjection());
              ShadowCameraTransformshadowCameraTransform = m_activeLight->
  CalcShadowCameraTransform(mainCamera.GetTransform().GetTransformedPos(),
                    mainCamera.GetTransform().GetTransformedRot());
              m_altCamera.GetTransform()->SetPos(shadowCameraTransform.GetPos());
              m_altCamera.GetTransform()->SetRot(shadowCameraTransform.GetRot());

              m_lightMatrix= BIAS_MATRIX *m_altCamera.GetViewProjection();

              SetFloat("shadowVarianceMin", shadowInfo.GetMinVariance());
              SetFloat("shadowLightBleedingReduction", shadowInfo.GetLightBleedReductionAmount());
              boolflipFaces = shadowInfo.GetFlipFaces();

        //const Camera* temp = m_mainCamera;
        //m_mainCamera = m_altCamera;

              if(flipFaces)
              {
                    glCullFace(GL_FRONT);
              }

              object.RenderAll(m_shadowMapShader, *this, m_altCamera);

              if(flipFaces)
              {
                    glCullFace(GL_BACK);
              }

        //m_mainCamera = temp;
              floatshadowSoftness = shadowInfo.GetShadowSoftness();
              if(shadowSoftness != 0)
              {
                    BlurShadowMap(shadowMapIndex, shadowSoftness);
              }
        }
```

```
else
        {
            m_lightMatrix=Matrix4f().InitScale(Vector3f(0,0,0));
            SetFloat("shadowVarianceMin", 0.00002f);
            SetFloat("shadowLightBleedingReduction", 0.0f);
        }

        GetTexture("displayTexture").BindAsRenderTarget();

        //m_window->BindAsRenderTarget();
        //glEnable(GL_SCISSOR_TEST);
        //TODO: Make use of scissor test to limit light area
        //glScissor(0, 0, 100, 100);

        glEnable(GL_BLEND);
        glBlendFunc(GL_ONE, GL_ONE);
        glDepthMask(GL_FALSE);
        glDepthFunc(GL_EQUAL);

        object.RenderAll(m_activeLight->GetShader(), *this, mainCamera);

        glDepthMask(GL_TRUE);
        glDepthFunc(GL_LESS);
        glDisable(GL_BLEND);

        //glDisable(GL_SCISSOR_TEST);
  }
  SetVector3f("inverseFilterTextureSize", Vector3f(1.0f/GetTexture("displayTexture").GetWidth(),
1.0f/GetTexture("displayTexture").GetHeight(), 0.0f));
  ApplyFilter(m_fxaaFilter, GetTexture("displayTexture"), 0);
}
```

Function AddLight()

```
inlinevoidAddLight(constBaseLight&light) { m_lights.push_back(&light); }
```

FunctionAddCamera()

```
inlinevoidAddCamera(constCamera&camera) { m_mainCamera = &camera; }
```

4. Lighting

Four types of lighting are used:

a. Base Light

The base light is ambient lighting. Ambient lighting is used to produce basic colours for objects. The light used in ambient lighting illuminates objects in all directions.

b. Directional Light

Directional Light is lighting where the light source is located at a point that is far away. Thus, the direction or vector of light to the object is parallel.

c. Point Light

Point light is the lighting effect where the light source is located at a certainpoint. The point light has a certain position andintensity.

d. Spot Light

Spot Light is almost similar to Point Light, the difference is that in spot light there is a fall off actor where a beam of light will appear brighter at a certain point then gradually loses it slight intensity when away from that point

5. Material

Two types of components are used in the material:

a. Texture

Texture is an image in which iti sattached to a certain object.Texture can be a regular image or a normal map

b. Lighting

Lighting is a lighting effect that has been previously described. There are several types of lighting used, namely base light, direction allight, pointlight and spotlight

6. Mesh

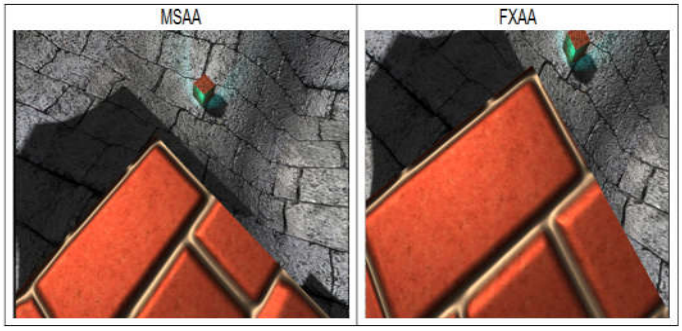
Mesh is the geometry of the object to berendered in the scene. The mesh has several components.

a. Vertex Coordinate
b. Texture
c. Normals
d. Colors

RESULTS AND DISCUSSION

After implementing two types of algorithms, namely MSAA and FXAA, we can measure three types of performance parameters, namely, the time it takes to render one frame, frames per second, and the quality of the resulting edge. The following are the results obtained when the authors compared the performances of MSAA and FXAA.

Anti Aliasing Aliasing	Milliseconds per Frame	FPS (Frame Per Seconds)
MSAA	40-50 ms	30-40 FPS
FXAA	16-20 ms	50-60 FPS



From this, we can see that, in terms of quality, FXAA and MSAA are not significantly different. However, from the performan cemetrics, namely milliseconds per frameor FPS, FXAA is able to perform the anti-aliasing process faster than MSAA.

CONCLUSION

In this study, the authors compare the performance and results of two anti-aliasing algorithms, namely MSAA and FXAA. From the results above, we can conclude that FXAA is better than MSAA because it is able to render images faster than MSAA. We can see that FXAA has a 30-50% faster rendering time than MSAA.

REFERENCES

1. Catmull, E. Hidden-surface algorithm with ant aliasing. Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, page 11. ACM. 1978
2. Herubel, Adrian., Biri, Venceslas. Morphological anti-aliasing and topological reconstruction. 2011
3. Reshetov and Alexander reduced aliasingarte facts by resampling. 2012
4. Auzinger, Thomas. Musialski, Przemyslaw., Preiner, Reinhold., Wimmer, Michael. Non Sampled Anti Aliasing. The Eurographic Association. 2013
5. Akeley, K. Reality Engine Graphics. In SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, Pages 109-116, New York. ACM

****