CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Uma solução baseada em Shader para renderizar um Mapa de Calor em um objeto 3D

LUCIANO SOARES BORELLI

Marília, 2016

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Uma solução baseada em Shader para renderizar um Mapa de Calor em um objeto 3D

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação
Orientador: Prof. Me. Allan Cesar Moreira de Oliveira

Marília, 2016

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Luciano Soares Borelli

Uma solução baseada em Shader para renderizar um Mapa de Calor em um objeto 3D.

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9.0 (NOVE)

Orientador:Allan Cesar Moreira de Oliveira_____

1°.Examinador:Leonardo Castro Botega_____

2°.Examinador:Jorge Luiz Barbosa Maciel Junior_____

Marília, 05 de dezembro de 2016.

Dedico este trabalho aos meus pais, minha esposa, meus irmãos, toda minha família, meus amigos e aos professores que me motivaram quando mais precisei.

AGRADECIMENTOS

Em primeiro lugar quero agradecer a Deus por estar do meu lado desde o primeiro dia em que iniciei a graduação, agradeço também por ele me apresentar soluções que foram essenciais para vencer as dificuldades que apareceram durante essa jornada.

Agradeço também a meus pais Gino e Rita, a minha esposa Jesuene, aos meus irmãos Thiago, Rafael, Eduardo e Isabella e a toda família, que me apoiou, incentivou e principalmente, acreditou que eu era capaz e graças a Deus, consegui.

Agradeço a todos os amigos e colegas de faculdade, agradeço pela oportunidade de aprender com todos os professores durante a graduação. E por fim, agradeço ao professor Allan por aceitar ser o orientador do meu trabalho de conclusão.

“Bendito seja o Senhor, minha rocha, que adestra as minhas mãos para a peleja e os meus dedos para a guerra.”
Sl 144:1

RESUMO

O processo de testes sobre automóveis, aeronaves ou qualquer equipamento é extremamente importante, a fim de evitar problemas que podem ser ocasionados por falhas humanas, mecânicas ou de software. Porém a aplicação dos testes realizados muitas vezes permite que alguma falha passe despercebida, ocasionando sérios riscos aos seus utilizadores. A apresentação dos resultados destes testes é exibida, na maioria das vezes, de forma abstrata, ou seja, apenas são apresentadas letras e números. Com estas informações, os responsáveis pelos testes podem não interpretar corretamente um resultado, ignorando, por exemplo, o fato de que a turbina de um determinado avião pode suportar até 1700º C, enquanto o resultado obtido durante os testes relatam que a temperatura é de 2000º C, podendo ocasionar uma grande catástrofe no momento em que o avião está em uso. Utilizando o conceito da Teoria do Ajuste Cognitivo de Iris Vessey, que explica a diferença entre os desempenhos de pessoas que utilizam informações apresentadas de maneira gráfica e das pessoas que utilizam informações apresentadas por meio de textos, entendemos que uma possível solução para reduzir o risco de falhas passarem despercebidas é facilitando a apresentação dos resultados, traduzindo as informações em texto por informações visuais. O projeto foi proposto a fim de oferecer uma solução que apresente um efeito de mapa de calor sobre um modelo 3D, sendo possível aplicar o mesmo em qualquer trabalho que dependa da visualização e compreensão da temperatura de equipamentos, assim como será possível apresentar os resultados obtidos nos testes de modo visual, permitindo que o responsável pelo teste possa ter um melhor desempenho e como conseqüência, reduza o risco de que possíveis falhas sejam ignoradas. Este projeto permite reproduzir temperaturas reais em modelos 3D. Para tornar isto possível utilizamos Shaders, a fim de sobrepor uma determinada região no Mapa UV dos modelos 3D.

Palavras-chave: Mapa de calor, Unity, Shader, ShaderLab, C#, Mapa UV, Internet das coisas, JSON, Modelo 3D, Ajuste Cognitivo.

ABSTRACT

The testing process on automobiles, aircraft or any equipment is extremely important in order to avoid problems that may be caused by human, mechanical or software failures. However, the application of the tests performed many times allows some failure to go unnoticed, causing serious risks to its users. The presentation of the results of these tests is shown, in most cases, in an abstract way, that is, only letters and numbers are displayed. With this information, testers may not correctly interpret a result, ignoring, for example, the fact that the turbine of a given aircraft can withstand up to 1700 ° C, while the result obtained during the tests report that the temperature is 2000 ° C, and can cause a major catastrophe at the time the aircraft is in use. Using the concept of Iris Vessey's Theory of Cognitive Adjustment, which explains the difference between the performances of people who use graphically presented information and the people who use information presented through texts, we understand that a possible solution to reduce the risk of Failures go unnoticed is facilitating the presentation of results, translating information into text for visual information. The project was proposed in order to offer a solution that presents a heat map effect on a 3D model, being possible to apply the same in any work that depends on the visualization and understanding of the equipment temperature, as well as to present the obtained results In visual mode testing, allowing the test taker to perform better and as a consequence, reduce the risk that failure should be ignored. This project allows you to reproduce real temperatures in 3D models. To make this possible use Shaders, an order to overlap a certain region in the UV Map of 3D models.

Keywords: Heat map, Unity, Shader, ShaderLab, C #, UV Map, Internet of things, JSON, 3D model, Cognitive fit.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2D	Bidimensional
3D	Tridimensional
IOT	Internet of Things (Internet das Coisas)
JSON	JavaScript Object Notation
XML	Extensible Markup Language

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

No início da década de noventa, Iris Vessey propôs uma teoria, nomeada Cognitive fit theory (Teoria do Ajuste Cognitivo), que descreve e explica o processo de conhecimento da cognição humana. Vessey explica através de sua teoria o desempenho diferenciado entre pessoas que utilizam informações apresentadas de maneira gráfica e pessoas que utilizam informações apresentadas por meio de textos.

A teoria foi utilizada em trabalhos posteriores da própria Vessey (1991, 1994 e 2006), sempre explicando os resultados de estudos publicados que analisam os diferentes desempenhos de acordo com a maneira em que as informações são apresentadas. Com base nos estudos de Vessey, é possível afirmar que a maneira em que as informações são apresentadas define o desempenho de quem as utiliza.

Pensando nos riscos que podem ser gerados por falhas e/ou problemas em equipamentos que passam despercebidos durante os testes em equipamentos ou mesmo que possam ocorrer durante a utilização de um equipamento nas residências, empresas ou mesmo durante uma viagem/passeio, entendemos que pode ter ocorrido uma falha na interpretação de um determinado resultado, possivelmente este foi apresentado de maneira abstrata, ou seja, através de letras e números ao responsável pelo teste, dificultando sua análise e entendimento.

Se para este problema aplicarmos a Teoria do Ajuste Cognitivo, pode-se deduzir que a interpretação dos resultados será mais bem entendida e conseqüentemente, será identificada a falha, reduzindo assim, o impacto que um possível problema teria.

O trabalho foi proposto a fim criar uma solução que capture em tempo real a temperatura de um determinado equipamento e represente fielmente essa informação de modo visual, gerando um efeito de mapa de calor.

As informações enviadas pelos dispositivos (utilizando o conceito da Internet das Coisas) são recebidas no mundo virtual, para que um programa (script) possa processar e enviar essa informação ao Shader, script que é processado na GPU, gerando efeitos sobre um modelo 3D para que no fim a cor do objeto seja sobreposta por uma cor abstraída da temperatura.

Figura 1 – Processo do Projeto

Fonte: Elaborado pelo Autor

Um exemplo seria um avião, que teve uma pane identificada por causa de um superaquecimento na hélice. Este projeto irá representar essa informação em tempo real, sobrepondo a cor cinza/preta da hélice pela cor vermelha. Com isto, será muito mais fácil e rápido de identificar aonde ocorreu o problema.

A solução proposta auxilia não só a apresentar de modo visual o que está ocorrendo com a hélice, mas sim com várias (N) partes do avião, em paralelo. Assim, será possível identificar de forma mais rápida que ao mesmo tempo em que a hélice está superaquecida, o Aileron (responsável por fazer o avião virar para os lados) da asa direita está congelado, impedindo que o avião faça uma determinada ação. Com isto, o próprio piloto visualizará as informações em tempo real; assim como um técnico de manutenção force diversos testes no equipamento (teste de stress) e identifique sobreaquecimentos de peças antes do avião ser liberado para vôo.

1.1 – Problemas

Para que o projeto fosse implementado, nos deparamos com alguns problemas que deviam ser solucionados. O primeiro problema foi como realizar a integração utilizando os conceitos da IoT com o Shader, visto que scripts em Shader não possuem tantas funções como scripts e Javascript e/ou C#.

Outro problema encontrado era como seria possível o Shader representar mais de uma região com informações diferentes, por exemplo: existem três sensores, cada sensor está em uma posição na vertical/horizontal do mapa UV do modelo 3D. Como seria possível distinguir essas três regiões diferentes e por fim, como fazer com que cada região tivesse uma cor diferente sobreposta, visto que a região no sensor um, pode estar com baixa temperatura, mas a região do sensor dois pode estar superaquecida.

1.2 – Motivação

Redução dos riscos que equipamentos possam vir a ter falhas ocasionadas por interpretações incorretas que foram realizadas por pessoas que não entenderam bem até onde é o limite de uma determinada parte e/ou peça, por causa da maneira em que as informações foram apresentadas para ele naquele momento.

Aplicando o conceito da Teoria do Ajuste Cognitivo, alterando a forma em que as informações apresentadas deixaram de ser apenas tabelas ou textos e passarão a ser representadas de modo visual, por meio de objetos 3D, permitiremos que os responsáveis por estes testes possam interpretar e entender melhor se aquele resultado realmente é o esperado.

Pensando em uma pessoa que está aplicando testes em um avião, esta pessoa pode muito bem verificar que a temperatura de uma das partes do avião está em 1800º C, porém por não entender a tabela que lhe foi apresentada, ela pode simplesmente interpretar que aquela temperatura esta dentro dos padrões estipulados, porém acontece que aquela peça suporta apenas 1750º C e que se por acaso esta temperatura passar dos 1850º C, a mesma pode sofrer um superaquecimento e com isto, gerar um incêndio.

Se houver uma maneira que permita esta pessoa de visualizar três modelos do mesmo equipamento, o risco que pode ser gerado por causa de um superaquecimento será mais bem compreendido e melhor interpretado: primeiramente é apresentado um modelo 3D do avião (ou qualquer outro equipamento/veículo) com um mapa de calor aplicado, apresentando as cores que representam a temperatura esperado para cada sensor; posteriormente é apresentado um segundo modelo (idêntico ao primeiro) com um mapa de calor aplicado que apresenta as cores da temperatura atual de cada sensor; por fim, é apresentado o terceiro modelo (idêntico ao primeiro) com um mapa de calor que apresenta a diferença entre a temperatura esperada e a temperatura atual de cada sensor, mostrando se aquela região está mais quente (ou mais fria) que o esperado. Conseqüentemente, estas melhores interpretações reduzirão os riscos que estes equipamentos possam vir a gerar para todos aqueles que o utilizarem, evitando possíveis acidentes, como por exemplo, a peça de um avião que pega fogo por não resistirem a uma temperatura maior que 1800º C.

A principal motivação deste projeto foi oferecer uma solução que auxilie as empresas em testes de stress durante a manutenção preditiva, para que seja possível identificar possíveis problemas com superaquecimentos em tempo real e melhor ainda, que essa informação seja visualmente representada por um modelo 3D do avião com as regiões quentes identificadas

pela cor vermelha e as regiões frias identificadas pela cor azul, gerando um efeito de mapa de calor sobre aquela representação 3D do avião naquele instante. Cada parte importante do avião terá um sensor e este, através da IoT, enviará informações sobre aquela região específica. Paralelamente a isso, estaremos contribuindo com os avanços na representação fiel do mundo real no mundo virtual.

A justificativa deste trabalho é permitir que tomadas de decisão sejam realizadas de forma ágil e confiável, graças a Internet das Coisas (IoT), que permite transmitir informações exatas de modo rápido e seguro. A agilidade se deve também graças a representação do Modelo 3D com seu mapa de calor, a fim de apresentar a visualização em tempo real da temperatura das partes.

É importante ressaltar que este projeto poderá ser usado não apenas na aviação, mas também em automóveis, geradores de hidroelétricas e qualquer outro equipamento que tenha problemas de superaquecimento e seja monitorado constantemente. Este projeto pode ser usado em diferentes segmentos da manutenção (ou outro domínio) que necessitem de informações em tempo real de temperaturas a fim de tomar a melhor decisão em um menor tempo, como por exemplo, identificar um possível risco ocasionado por um superaquecimento de uma sala de servidores e conseqüentemente, tomar a melhor decisão para reduzir o impacto daquela ocorrência.

1.3 – Objetivos

O objetivo geral do projeto é representar a temperatura exata de um determinado objeto (do mundo real) naquele determinado instante em um modelo 3D correspondente.

Um dos objetivos específicos é criar um método de mapeamento dos sensores em suas determinadas regiões no modelo 3D, representando fielmente o objeto do mundo real.

Outro objetivo é gerar um efeito de mapa de calor sobre o modelo 3D, permitindo identificar de forma rápida uma possível falha e/ou problema. Este mapa de calor deve ser gerado utilizando informações formatadas em um arquivo json, permitindo que qualquer serviço atual que utilize o mesmo padrão possa fazer a requisição destes dados.

1.4 – Metodologia

Para a realização do projeto foi necessário primeiramente entender como funcionam os modelos 3D, desde o momento em que ele é criado até a geração do Mapa UV e sua integração com o modelo 3D.

Após conhecer melhor os modelos 3D, é necessário conhecer melhor pipeline de renderização, em particular o Shader e sua função.

Também foi necessário conhecer um pouco mais sobre a Internet das Coisas (IoT) e saber mais sobre as melhores formas de troca de informação, entre elas o padrão JSON.

Por fim, a última etapa foi realizar a integração de todas as etapas anteriores, permitindo a implementação do projeto.

1.5 – Estrutura do Trabalho

O projeto foi segmentado em quatro partes: a primeira explica o que são Shaders, como funcionam, os tipos e como o mesmo pode ser utilizado na Unity. A segunda parte explica detalhadamente como o projeto foi elaborado e desenvolvido em todas suas etapas. Posteriormente é apresentado um estudo de caso e um complemento da solução proposta. Por fim, a última parte apresenta a conclusão do projeto, os resultados obtidos, as limitações e sugestões de continuidade deste trabalho.

CAPÍTULO 2 – SHADERS

Com o modelo 3D pronto, é possível controlar o comportamento e/ou aplicar efeitos sobre as superfícies dos objetos através de Shaders. O processo de Shading utiliza uma equação para computar o comportamento da superfície de um objeto. (AKENINE-MÖLLER, 2008)

O comportamento é programado através de Shaders, que nada mais são do que algoritmos/instruções enviadas a GPU (unidade de processamento gráfico) através de scripts que podem ser escritos utilizando linguagens de programação como CG, GLSL, ShaderLab, entre outros.

Shaders são os responsáveis por gerar efeitos sobre um objeto, como por exemplo: gerar o efeito de ondas de rios/mares, apresentar o reflexo de objetos próximos, deixarem as superfícies com o efeito de molhadas após entrar em contato com água, dar o efeito metálico nas carcaças de veículos, entre outros. É possível aplicar diversos Shaders em um único objeto, dando a possibilidade de que cada Shader seja responsável por uma parte especifica do objeto. (PASSOS, 2009)

2.1 – Funcionamento

Para que um Shader possa gerar o efeito desejado sobre um objeto 3D, é necessário que o objeto possua um Material, que nada mais é do que um container para as propriedades visuais do objeto.

Um vértice é primariamente um ponto simples ou posição no espaço 3D. Um fragmento é um conjunto de dados gerado pela rasterização, como coordenadas, profundidade, cor e coordenada da textura. (EVANGELISTA, 2010)

O Shader é responsável por definir o método de renderização do objeto e as propriedades, assim como define também todos os Vertex e Fragment Shaders utilizados na renderização. Já o Material é responsável por definir os valores a serem utilizadas nas propriedades, como a textura, valores/números e as cores utilizadas para a renderização. (PASSOS, 2009)

2.2 – Tipos de Shaders

Entre os principais tipos de Shaders, destaca-se:

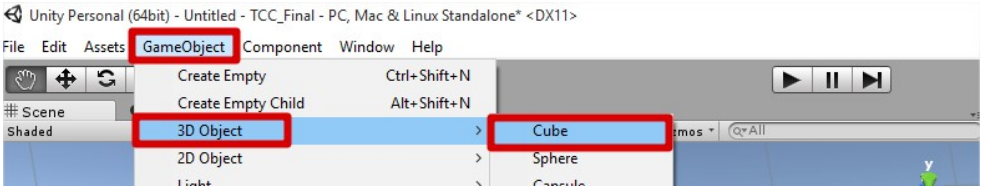
- Vertex Shader: função que através de operações matemáticas permite manipular os vértices do objeto, alterando a coordenada de posição do mesmo de uma maneira mais fácil e ágil. Com isto, é possível transformar a posição do vértice, iluminar um vértice, gerar as coordenadas de textura para a posição do vértice e determinar o material a ser aplicado ao vértice. Como parâmetro de entrada a função do Vertex Shader requer um vértice e no fim retorna o mesmo vértice, porém com as modificações realizadas durante seu processamento. (FEIJÓ, 2006)

- Fragment Shader: função que opera na etapa de processamento dos fragmentos e atribui uma cor para cada fragmento. Com isto, é possível definir qual a cor será apresentada sobre o fragmento correspondente daquela superfície. Como parâmetro de entrada a função do Fragment Shader requer um fragmento e no fim retorna a cor a ser apresentada no pixel. (LIMA, 2014)

2.3 – Utilização de Shader na Unity

Antes de iniciar a utilizar Shader na Unity, é necessário primeiramente importar/criar um objeto e adicionar o mesmo na cena. Para adicionar um modelo 3D na forma básica, acesse “GameObject” > “3D Object” > “Cube”, localizado no menu do programa.

Figura 2 – Adicionando um objeto 3D na cena (Unity)

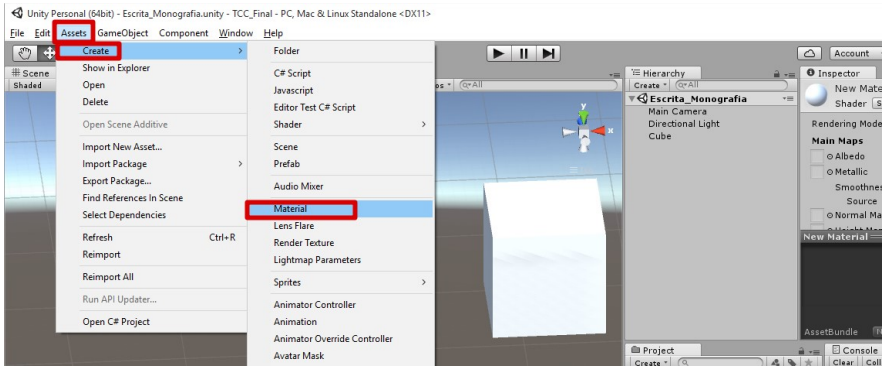


Fonte: Elaborado pelo Autor

Após adicionar o objeto 3D na cena, você precisa verificar se o mesmo já possui um

Material vinculado. Para isto, clique sobre o objeto na cena ou selecione o objeto na aba “Hierarchy” – listagem com os objetos/game objects. Ao selecionar, a aba “Inspector” apresentará todos os componentes e propriedades deste objeto. No caso, precisamos acessar o componente “Mesh Renderer” e verificar se na propriedade “Materials” existe algum Material definido. Caso esteja “Default-Material”, será necessário criar um Material. Para isto, acesse “Assets” > “Create” > “Material” no menu do programa. O Material recém-criado aparecerá na aba “Project”, que apresenta o diretório completo da pasta “Assets” do projeto.

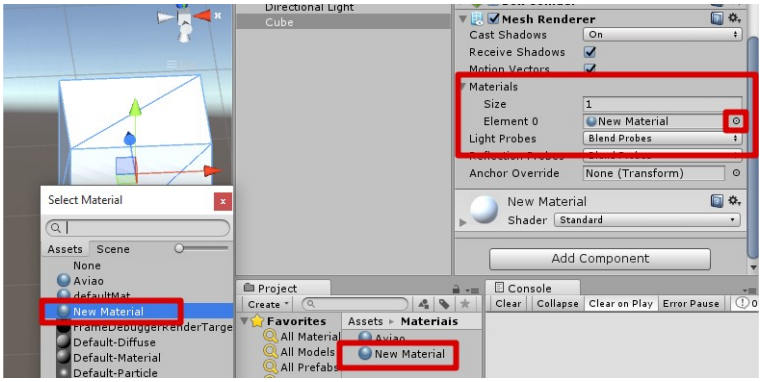
Figura 3 – Adicionando um Material nos Assets (Unity)



Fonte: Elaborado pelo Autor

Após criar o Material, será necessário definir o mesmo na propriedade. Clique sobre o objeto 3D, posteriormente acesse o componente “Mesh Renderer” na aba “Inspector” e por fim, clique no botão destacado na figura abaixo para selecionar o material recém-criado.

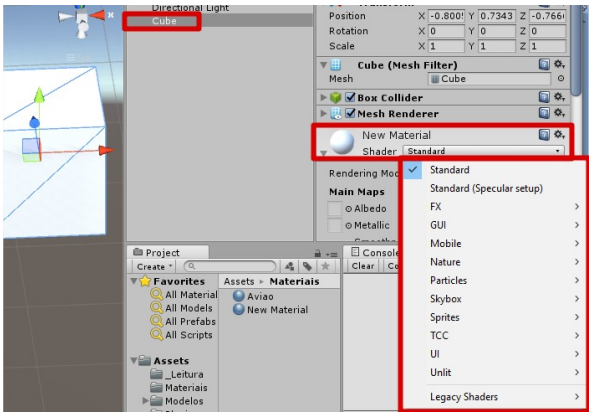
Figura 4 – Definindo um Material no Objeto 3D (Unity)



Fonte: Elaborado pelo Autor

Com o novo material atrelado ao componente “Mesh Renderer” do objeto 3D, agora é possível vincular um Shader ao mesmo, para que o efeito seja gerado no momento da execução. É possível alterar o Shader do material diretamente no objeto 3D ou acessando as propriedades do material através da aba “Project”.

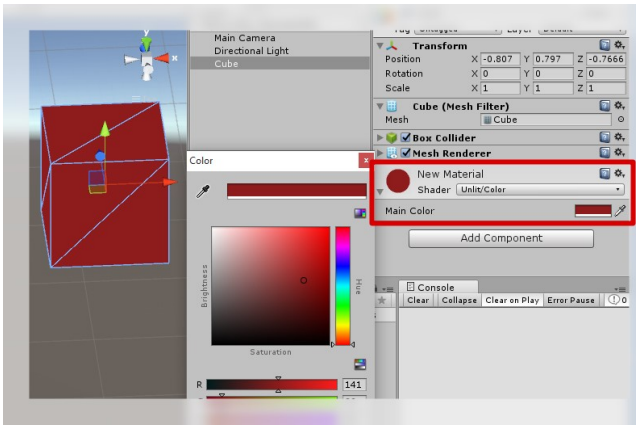
Figura 5 – Definindo o Shader no Material (Unity)



Fonte: Elaborado pelo Autor

A Unity oferece um conjunto de Shaders a serem usados, cada um com sua funcionalidade. Para conhecer um pouco sobre o que é um Shader, selecione o grupo “Unlit” e o script “Color”. Serão apresentadas as propriedades a serem definidas deste Shader, neste caso é possível definir a cor que irá sobrepor o objeto:

Figura 6 – Definindo a(s) propriedade(s) de um Shader através da aba “Inspector” (Unity)

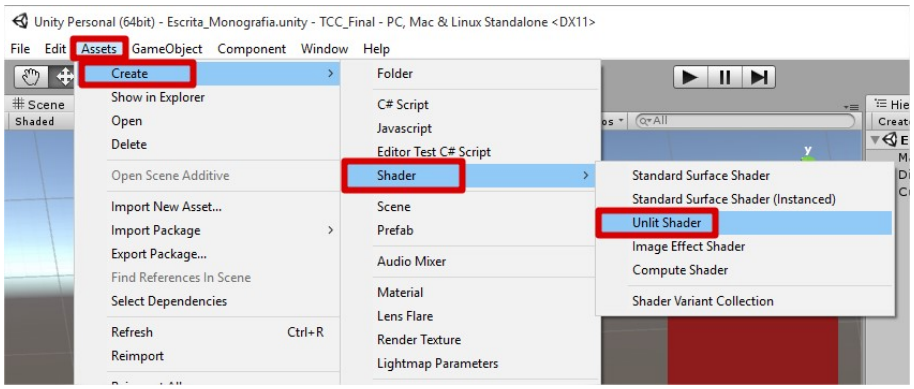


Fonte: Elaborado pelo Autor

Agora que entendemos o que o Shader pode fazer e onde podemos definir as propriedades do mesmo, vamos agora criar um novo Shader e conhecer um pouco a estrutura de um script na linguagem de programação ShaderLab, utilizada na Unity. Acesse no menu a opção “Assets” > “Create” > “Shader” > “Unlit Shader”.

.

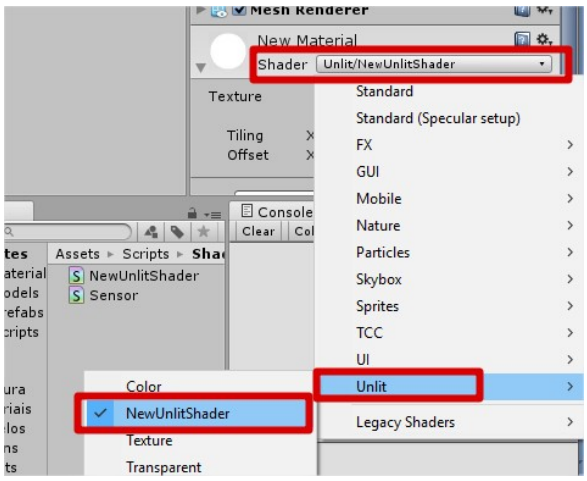
Figura 7 – Adicionando um Shader nos Assets (Unity)



Fonte: Elaborado pelo Autor

Com o script criado, agora é necessário vincular o mesmo no Material, para isto você pode clicar no objeto 3D e na aba “Inspector” você altera o Shader para o recém criado. O mesmo aparecerá no grupo “Unlit”:

Figura 8 – Alterando o Shader do Objeto (Unity)



Fonte: Elaborado pelo Autor

Para visualizar o conteúdo do script, localize o mesmo na aba “Projects” e de um

duplo clique para iniciar a edição.

A estrutura de um script Shader em ShaderLab é bem simples, permitindo que um mesmo Shader tenha várias propriedades e vários “SubShaders”. Cada SubShader pode ter uma coleção de um ou mais “passos” de execução.

Para iniciar o script em ShaderLab, primeiramente é necessário definir o grupo e o nome do Shader à ser criado. Logo em seguida, é possível definir as propriedades do Shader, que podem ter seus valores manipulados/alterados através das propriedades do Material. No exemplo abaixo, foram criados dois atributos que possuem o tipo Color, que nada mais é do que o conjunto RGBA (Red,Green,Blue,Alpha), onde cada número deste conjunto pode variar de zero a um e um atributo que possui o tipo Texture2D. Por padrão é definido uma textura em branco, porém é possível atribuir aqui uma textura desejada, como por exemplo, uma imagem:

```
Shader "Pintura/1Shader" {
	Properties {
		_corPadrao ("Cor Padrao", Color) = (1,0,0,1)
		_corSobreposicao ("Cor Sobreposicao", Color) = (0,0,1,1)
		_texturaPadrao ("Textura Padrao", 2D) = "white" {}
	}

	// continuação do código abaixo....
	// subshader { ... }
}
```

Como informado anteriormente, um Shader pode ter uma coleção de SubShaders. Quando o Unity tem que exibir uma malha, ele irá escolher o primeiro SubShader que é executado na placa gráfica do usuário. Os SubShaders utilizam tags para dizer como e quando eles esperam ser renderizados para o mecanismo de renderização. No exemplo abaixo é utilizado a tag “RenderType” que define um tipo de renderização opaca e é definido o nível 100 para o LOD (Shader Level of Detail). O LOD permite que você não utilize o mapeamento normal do paralaxe em placas gráficas de baixo custo:

```
SubShader {
        Tags { "RenderType"="Opaque" }
        LOD 100

        // continuação do código abaixo....
        // pass { ... }
}
```

Um SubShader pode ter vários “Passos”. Um passo faz com que a geometria de um GameObject seja renderizada uma vez. O código de programação do Shader é iniciado pelo comando CGPROGRAM e encerrado pelo comando ENDCG. Logo em seguida, é necessário definir as diretivas das funções. No exemplo abaixo a função Vertex Shader é nomeada como “funcVertives” e a função Fragment Shader chamada de “funcFragmentos”. Posteriormente é incluído um arquivo chamado “UnityCG.cginc” que contém variáveis predefinidas e funções auxiliares:

```
Pass {
        CGPROGRAM

        #pragma vertex funcVertices
        #pragma fragment funcFragmentos

        #include "UnityCG.cginc"

        // continuação do código abaixo....
        // struct Vertices { ... }

        ENDCG
}
```

Após definir as diretivas, o próximo passo é criar uma estrutura de dados que receba as coordenadas e a posição do vértice. Definimos como atributos a primeira coordenada UV através da palavra TEXCOORD0 e a posição através da palavra SV_POSITION. Com isto será possível saber exatamente a posição XY do vértice e a posição do objeto no espaço. Posteriormente, é necessário declarar as variáveis a serem utilizadas, no exemplo abaixo declaramos uma variável que recebe as propriedades da textura 2D, uma variável com o mesmo nome da variável anterior, porém com a inclusão do termo “_ST” ao fim do nome do

tipo Float4 (conjunto de quatro números) que receberá as propriedades Tiling e Offset da textura. Por fim, são criadas duas variáveis que recebem os valores atribuídos nas propriedades:

```
struct Vertices {
        float2 coordenada : TEXCOORD0;
        float4 posicao : SV_POSITION;
};

sampler2D _texturaPadrao;
float4 _texturaPadrao_ST;
fixed4 _corPadrao;
fixed4 _corSobreposicao;

// continuação do código abaixo....
// Vertices funcVertices ( ... ) { ... }
```

Posteriormente é necessário programar a função do Vertex Shader. No exemplo abaixo a função é chamada funcVertices, o mesmo nome declarado no momento da definição da diretiva do Vertex Shader. Como parâmetro de entrada, a função recebe as posições e coordenadas dos vértices. É possível também receber outras informações como parâmetro de entrada, como por exemplo, a cor da face daquele vértice. Porém é importante ressaltar que para a inclusão de outros parâmetros, deve-se criar um atributo na estrutura criada anteriormente.

Será necessário instanciar um objeto com a estrutura criada e definir seus atributos. No atributo “posição” é definida a multiplicação do modelo com a matriz de projeção e no atributo “coordenada” é utilizada a macro TRANSFORM_TEX do arquivo UnityCG.cginc para certificar-se de que a escala e o deslocamento da textura sejam aplicados corretamente. A função retornará a estrutura com as informações atualizadas:

```
Vertices funcVertices (float4 pPosicao : POSITION, float4 pCoordenada : TEXCOORD0) {
        Vertices estrutura;

        estrutura.posicao      = mul(UNITY_MATRIX_MVP, pPosicao);
        estrutura.coordenada  = TRANSFORM_TEX(pCoordenada, _texturaPadrao);

        return estrutura;
```

```
}

// continuação do código abaixo....
// fixed4 funcFragmentos ( ... ) { ... }
```

O próximo processamento é realizado através do Fragment Shader. Assim como a Vertex Shader, o nome da função é a mesma declarada no momento da definição da diretiva do Fragment Shader. Como parâmetro de entrada é atributo a estrutura processada pela Vertex Shader. No exemplo abaixo é criada uma variável do tipo Fixed4 (cor RGBA de baixa precisão) e um teste é realizado: Se a posição X da coordenada do vértice for menor ou equivalente a um determinado número, a variável receberá o mesmo valor que foi definido na variável "_corPadrao". Se o resultado do teste não for verdadeiro, o valor a ser definido é o mesmo da variável "_corSobreposicao". É importante destacar que a posição X e/ou Y possuem como valor um número de zero a um. Por retornar a cor no formato Fixed4, é necessário informar no momento da declaração da função que a saída possui a semântica SV_TARGET:

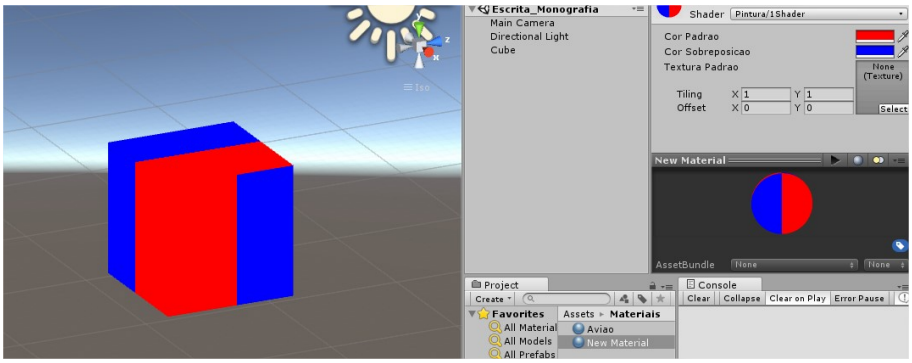
```
fixed4 funcFragmentos (Vertices estrutura) : SV_Target {
	fixed4 texturaSaida;

	if(estrutura.coordenada.x <= 0.55){
		texturaSaida = _corPadrao;
	}
	else{
		texturaSaida = _corSobreposicao;
	}

	return texturaSaida;
}
```

O conjunto do script acima irá verificar a posição X da textura padrão. Se a posição X for igual ou menor que 0,55 (um pouco mais que a metade), o Shader irá pintar aquela região de vermelho. Caso seja maior que 0,55, o Shader irá pintar aquela região restante de azul, conforme figura abaixo:

Figura 9 – Resultado do primeiro Shader (Unity)



Fonte: Elaborado pelo Autor

É possível também inserir uma textura no objeto, fazendo com que o mesmo possa apresentar a textura ao invés de cores definidas. Reutilizando todo o script acima, é necessário primeiramente alterar o valor “white” da propriedade da textura para o caminho em que se encontra a imagem da textura à partir da pasta “Assets” e/ou escolher a imagem diretamente pelo painel “Inspetor” da Unity, nas propriedades do material.

Logo em seguida, é necessário inserir um atributo na estrutura para receber a cor da face do vértice:

```
struct Vertices {
        float2 coordenada : TEXCOORD0;
        float4 posicao : SV_POSITION;
        float4 cor : COLOR;
};
```

No Vertex Shader é necessário incluir um parâmetro de entrada e a atribuição do mesmo na estrutura de dados:

```
Vertices funcVertices (float4 pPosicao : POSITION, float4 pCoordenada : TEXCOORD0 , float4 pCor : COLOR) {
        Vertices estrutura;

        estrutura.posicao = mul(UNITY_MATRIX_MVP, pPosicao);
        estrutura.coordenada  = TRANSFORM_TEX(pCoordenada, _texturaPadrao);
        estrutura.cor = pCor;


        return estrutura;
```

```
}
```

Por fim, para que as cores da textura sejam apresentadas no modelo 3D, basta incluir o trecho destacado abaixo no Fragment Shader para que seja retornada a cor exata da face daquele vértice:

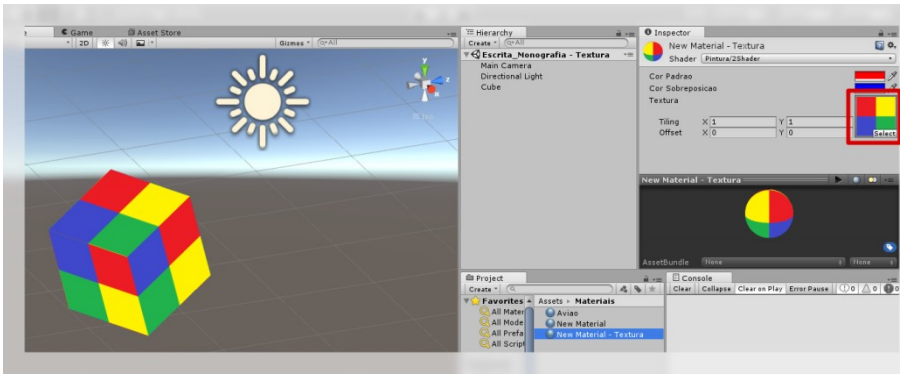
```
fixed4 funcFragmentos (Vertices estrutura) : SV_Target {
        fixed4 texturaSaida;

        texturaSaida = tex2D(_texturaPadrao,estrutura.coordenada) * estrutura.cor;

        return texturaSaida;
}
```

Com a utilização do script anterior e aplicada as modificações acima, teremos o seguinte resultado:

Figura 10 – Resultado do segundo Shader (Unity)



Fonte: Elaborado pelo Autor

Neste projeto, os Shaders terão papel fundamental para que seja possível identificar os pontos quentes e frios no objeto 3D.

CAPÍTULO 3 – UMA SOLUÇÃO BASEADA EM SHADER PARA RENDERIZAR UM MAPA DE CALOR EM UM OBJETO 3D

Com base na metodologia deste trabalho, a última etapa é realizar a integração de todos os conceitos envolvidos, permitindo a implementação do projeto e entendendo melhor como o projeto foi desenvolvido e como os problemas descritos na introdução são solucionados. Abaixo a figura ilustra que as etapas de entendimento do modelo 3D, dos Shaders e do conceito da Internet das Coisas já foram realizadas:

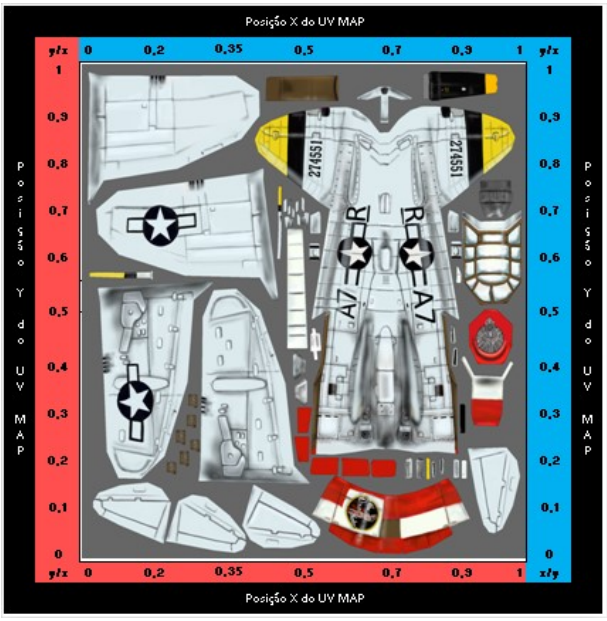
Figura 11 – Etapas concluídas do Projeto

Fonte: Elaborado pelo Autor

3.1 – Definição do posicionamento dos Sensores

Entendendo que as coordenadas dos modelos 3D variam de zero a um, foi necessário aplicar uma régua para conhecer as dimensões que uma determinada região é representada na textura. Na figura abaixo, podemos visualizar a UV Map com a textura aplicada e o posicionamento X e Y do mesmo:

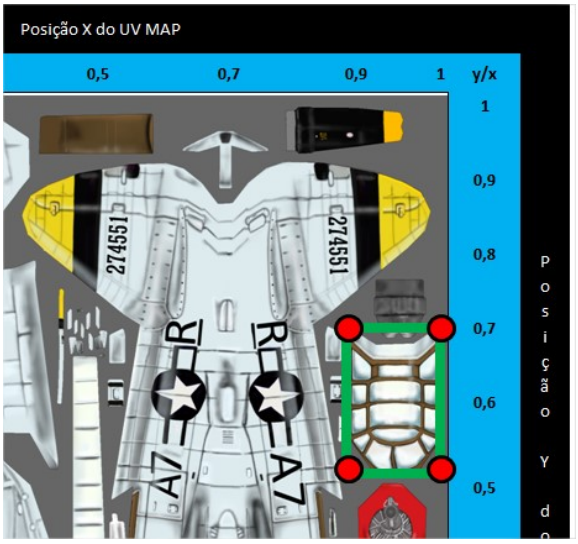
Figura 12 – Textura do Modelo 3D com a aplicação de uma “régua”



Fonte: Elaborado pelo Autor

Para que as cores correspondentes a temperatura sejam apresentadas precisamente na região correta no modelo 3D, a solução proposta é realizar o mapeamento dos componentes do avião. Para isto, é necessário definir qual é a posição inicial do ponto X, a posição inicial do ponto Y, a posição final do ponto X e a posição final do ponto Y. Abaixo segue um exemplo de como o posicionamento de um sensor pode ser realizado:

Figura 13 – Definição do posicionamento do Sensor da Cabine de um Avião



Fonte: Elaborado pelo Autor

Considerando o exemplo acima, é possível entender que o posicionamento da Cabine do avião fica entre os pontos 0,87 e 0,98 do eixo X e entre os pontos 0,57 e 0,70 do eixo Y. Assim, quando a temperatura estiver considerada “Muito Quente”, toda a região que estiver entre os pontos citados anteriormente terá a sobreposição da cor vermelha.

Foi elaborada uma tabela contendo os posicionamentos de alguns dos principais componentes do avião:

Tabela 1 – Posicionamento de alguns dos principais componentes do avião

Descrição	**Pos. Inicial X**	**Pos. Inicial Y**	**Pos. Final X**	**Pos. Final Y**
Cabine	0,87	0,57	0,98	0,68
Helice	0,80	0,93	0,95	1,00
Lado Direito do Estabilizado Vertical	0,80	0,79	1,00	0,91
Lado Esquerdo do Estabilizado Vertical	0,44	0,79	0,60	0,91
Lateral Direita da Fuselagem	0,70	0,20	0,90	0,70
Lateral Esquerda da Fuselagem	0,50	0,20	0,70	0,70
Pintura da Cabine	0,40	0,90	0,55	1,00

3.2 – Definição das estruturas do JSON

Após entender como o posicionamento dos sensores são feitos em relação a UV MAP, é possível definir o padrão a ser estabelecido para a declaração dos sensores que serão identificados no projeto.

Primeiramente foi necessário elaborar a estrutura para a declaração dos sensores e seus posicionamentos perante o UV MAP. Para isto, é definido que o rótulo “sensores” receberá um conjunto com no mínimo a declaração de um sensor. Cada sensor deve ter os seguintes atributos descritos:

- id: um número (inteiro) que identifique o sensor.
- Desc: uma descrição que contenha o nome da peça/parte referente aquele sensor.
- Pos_ini_x: um número (decimal) que contenha o posicionamento inicial da região daquele sensor no eixo X (horizontal).
- Pos_ini_y: um número (decimal) que contenha o posicionamento inicial da região

daquele sensor no eixo Y (vertical).

- Pos_fin_x: um número (decimal) que contenha o posicionamento final da região daquele sensor no eixo X (horizontal).
- Pos_fin_y: um número (decimal) que contenha o posicionamento final da região daquele sensor no eixo Y (vertical).

Abaixo é apresentado um exemplo do arquivo JSON contendo a declaração dos sensores e suas posições no modelo 3D:

```
{ "sensores":[
 { "id":"1", "desc":"Cabine", "pos_ini_x":"0.87", "pos_ini_y":"0.57", "pos_fin_x":"0.98", "pos_fin_y":"0.68" },
 { "id":"2", "desc":"Helice", "pos_ini_x":"0.8", "pos_ini_y":"0.93", "pos_fin_x":"0.95", "pos_fin_y":"1" }
] }
```

A outra estrutura foi criada para receber as informações sobre a temperatura em um determinado instante de cada sensor. Foi definido que o rótulo “temperaturas” receberá um conjunto de com no mínimo um registro que contenha a temperatura. Cada registro deve ter os seguintes atributos descritos:

- id_sensor: um número (inteiro) que represente o sensor declarado anteriormente.
- data_hora: uma data (formato AAAA-MM-DD) e uma hora (formato HH:MM:SS) que represente o instante correspondente daquela informação. A data e a hora devem ser separadas por um espaço em branco “ ”.
- Temperatura Normal: um número (decimal), podendo ser negativo (indicado através do símbolo de menos “-”), representando a temperatura considerada normal ou esperada daquele sensor.
- Temperatura: um número (decimal), podendo ser negativo (indicado através do símbolo de menos “-”), representando a temperatura daquele sensor naquela data e

naquela hora.

Abaixo segue o exemplo da estrutura que recebe um conjunto de temperaturas:

```
{ "temperaturas":[
 {"id_sensor":"1", "data_hora":"2016-10-14 20:10:00", "temp_normal":"14", "temperatura":"27" },
 {"id_sensor":"2", "data_hora":"2016-10-14 20:10:00", "temp_normal":"25", "temperatura":"4" },
 {"id_sensor":"12", "data_hora":"2016-10-14 20:10:00", "temp_normal":"-5", "temperatura":"-8" },
 {"id_sensor":"1", "data_hora":"2016-10-14 20:11:00", "temp_normal":"14", "temperatura":"14" }
] }
```

O arquivo JSON com as declarações dos sensores deve ser nomeado como “sensores.json” e o arquivo contendo as temperaturas deve ser nomeado como “temperaturas.json”

3.3 – Elaboração do script Shader na Unity

O próximo passo é elaborar o script Shader para que os efeitos sejam aplicados sobre a textura do objeto 3D. O script foi nomeado como “Sensor.shader” e o Shader foi nomeado como “TCC/Sensor”. Primeiramente foram definidas as propriedades do Shader (Cor Fria, por padrão a cor azul clara; Cor Muito Fria, por padrão a cor azul escura; Cor Muito Quente, por padrão a cor vermelha; Cor Quente, por padrão a cor vermelha clara; e a Textura a ser aplicada no objeto, por padrão a imagem da textura correspondente ao UV Map do avião), conforme trecho abaixo:

```
Properties {
        _corTempFrio("Preenchimento Frio", Color) = (0.50,0.50,1.00,1)
        _corTempMuitoFrio("Preenchimento Muito Frio", Color) = (0.00,0.00,1.00,1)
        _corTempMuitoQuente("Preenchimento Muito Quente", Color) = (1.00,0.00,0.00,1)
        _corTempQuente("Preenchimento Quente", Color) = (1.00,0.50,0.50,1)
```

```
        _texturaPadrao("Textura", 2D) = "Modelos/DownNecker/done2.png" {}
}
```

Foi definida também a estrutura de dados que receberá as informações dos vértices, tendo um atributo para receber a primeira coordenada do vértice, outro para receber a posição do objeto e por fim outro para receber a cor correspondente aquele vértice:

```
struct EstruturaDados {
        float2 coordenada : TEXCOORD0;
        float4 posicao : SV_POSITION;
        fixed4 cor : COLOR;
};
```

Além das propriedades e da estrutura, é necessário declarar variáveis que podem ser acessadas através de outros scripts, como nosso próximo script, que é feito em C#. Além de declarar as propriedades como variáveis (incluindo a textura com a sigla "_ST" no final da nomenclatura), é declarada também uma variável do tipo "inteiro" que possui o número total de sensores que o projeto possui e quatro conjuntos (arrays): Posições Iniciais do eixo XY, Posições Finais do eixo XY, Temperaturas Normais e as Temperaturas daquele instante:

```
sampler2D _texturaPadrao;
float4 _texturaPadrao_ST;
fixed4 _corTempFrio;
fixed4 _corTempMuitoFrio;
fixed4 _corTempMuitoQuente;
fixed4 _corTempQuente;
int _qSensores = 0;
float2 _PosicoesIniciais[100];
float2 _PosicoesFinais[100];
float2 _TempNormal[100];
float2 _Temperaturas[100];
```

A linguagem ShaderLab, pelo menos até a versão 5.4.1f1 da Unity, possui uma

limitação em que não é possível criar arrays com seu tamanho variável, com isto, é necessário declarar um tamanho máximo para os arrays. Neste projeto foi definido o tamanho de 100 posições para os conjuntos (arrays), conforme trecho acima. Vale ressaltar que utilizar arrays em Shaders só é possível a partir da versão **5.4.1f1** da Unity.

A Vertex Shader recebe os parâmetros de entrada e aplica os mesmos na estrutura criada anteriormente. Por fim, a Fragment Shader recebe como parâmetro de entrada a estrutura processada na Vertex Shader e, após seu processamento retorna a cor desejada no fragmento.

Durante o processamento da Fragment Shader, é instanciada uma variável que recebe, por padrão a cor atual do vértice:

```
fixed4 texturaSaida = tex2D(_texturaPadrao,estrutura.coordenada) * estrutura.cor;
```

Posteriormente, é executado um loop, que tem como limite de voltas a quantidade de sensores na variável "_qSensores". A cada volta, é executado uma série de testes lógicos, sendo o primeiro para verificar se a coordenada XY atual se enquadra nas posições XY declaradas no sensor:

```
if(estrutura.coordenada.x >= _PosicoesIniciais[c].x && estrutura.coordenada.x <= _PosicoesFinais[c].x &&
estrutura.coordenada.y >= _PosicoesIniciais[c].y && estrutura.coordenada.y <= _PosicoesFinais[c].y){
```

Caso o teste retorne "verdadeiro", o próximo teste lógico tem como objetivo validar se a temperatura daquele sensor é menor que 0º. Se seu resultado for "verdadeiro", é atribuída a cor referente à variável "_corTempMuitoFrio" e o mesmo, por fim, é apresentado como a cor do fragmento:

```
if(_Temperaturas[c].x < 0){
        texturaSaida = _corTempMuitoFrio;
}
```

Caso o teste retorne "falso", o próximo teste lógico tem como objetivo validar se a temperatura daquele sensor é menor que 15º. Se seu resultado for "verdadeiro", é atribuída a cor referente à variável "_corTempFrio" e o mesmo, por fim, é apresentado como a cor do

fragmento:

```
if(_Temperaturas[c].x < 15){
        texturaSaida = _corTempFrio;
}
```

Se o teste retornar “falso”, o próximo teste lógico tem como objetivo validar se a temperatura daquele sensor é menor que 30º. Se seu resultado for “verdadeiro”, é atribuída a cor referente à variável “_corTempQuente” e o mesmo, por fim, é apresentado como a cor do fragmento:

```
if(_Temperaturas[c].x < 30){
        texturaSaida = _corTempQuente;
}
```

Por fim, se o teste retornar “falso”, o próximo e último teste lógico tem como objetivo validar se a temperatura daquele sensor é maior ou igual a 30º. Se seu resultado for “verdadeiro”, é atribuída a cor referente à variável “_corTempMuitoQuente” e o mesmo, por fim, é apresentado como a cor do fragmento:

```
if(_Temperaturas[c].x >= 30){
        texturaSaida = _corTempMuitoQuente;
}
```

Este script, como informando anteriormente, tem como objetivo sobrepor a cor da textura na região correspondente ao posicionamento do sensor de acordo com a temperatura do sensor, gerando o efeito de mapa de calor.

3.4 – Elaboração do script C# na Unity

Para resolver o problema de integração entre as informações vindas dos arquivos JSON com o Shader, é utilizado um script escrito em C#, que é atrelado ao GameObject, ou

seja, ao modelo 3D. O objetivo do script é obter as informações em tempo real e posteriormente, definir as variáveis do Shader, alterando as cores dos sensores de acordo com as informações de cada instante.

O script é nomeado como “Aviao.cs” e utiliza algumas das principais bibliotecas utilizadas em C#, bem como uma biblioteca chamada “SimpleJSON”, que está disponível para download através da página <http://wiki.unity3d.com/index.php/SimpleJSON>. Esta biblioteca foi necessária para que fosse possível, de um jeito prático e fácil, a utilização das informações vindas no formato JSON, principalmente no manuseio de arrays.

O script possui uma classe chamada “Aviao”, esta possui como atributo um conjunto (array) de sensores (classe que possui como atributos um identificador, uma descrição, as posições iniciais da coordenada XY, as posições finais das coordenadas XY e um conjunto (arrays) de temperaturas (classe que possui a data, hora, temperatura normal e a temperatura como atributos)), conforme trecho abaixo:

```
public class Aviao : MonoBehaviour {

	Sensor[] Sensores;

	public class Temperatura {

		public string data;

		public string hora;

		public float temp_normal;

		public float temperatura;

	}


	public class Sensor {

		public int id;

		public string desc;

		public float pos_ini_x;

		public float pos_ini_y;

		public float pos_fin_x;

		public float pos_fin_y;

		public Temperatura[] temperatura;
```

```
        }
```

Foi criada uma função para abrir e efetuar a leitura dos arquivos JSON. Chamada de "CarregarConteudoArquivo", esta função retorna o conteúdo dos arquivos JSON:

```
string CarregarConteudoArquivo(string nomearquivo) {
        string linhas = "";
        string retorno = "";

        StreamReader leitura = new StreamReader(nomearquivo, Encoding.Default);

        using(leitura) {
                do {
                        linhas = leitura.ReadLine();

                        if(linhas != null) {
                                retorno += linhas;
                        }
                } while(linhas != null);

                leitura.Close();

                return retorno;
        }
}
```

A execução do script, após a definição dos atributos e das funções, é iniciada no método "Start". Ao ser iniciado, o script faz a leitura do arquivo sensores.json. É importante registrar que no momento, é feito a leitura de um arquivo JSON já elaborado anteriormente, mas com a inclusão de uma função que faça a requisição para um determinado IP do sensor e obter esse arquivo diretamente através da Internet das Coisas. O script está preparado para receber esta implementação futuramente. Abaixo segue o trecho que realiza o armazenamento

do conteúdo dos arquivos e posteriormente desserializa o mesmo:

```
void Start() {
        string leituraSensores = "";
        string leituraTemperaturas = "";

        leituraSensores = CarregarConteudoArquivo("Assets/_Leitura/sensores.json");
        leituraTemperaturas = CarregarConteudoArquivo("Assets/_Leitura/temperaturas.json");

        var conteudoSensores = JSON.Parse(leituraSensores);
        var conteudoTemperaturas = JSON.Parse(leituraTemperaturas);

        Sensores = new Sensor[conteudoSensores["sensores"].AsArray.Count];
```

Na última linha do trecho acima, é definido o tamanho do conjunto de sensores através da função "AsArray.Count", que retorna a quantidade de elementos pertencentes aquele conjunto. Abaixo podemos ver que foi necessário criar um vetor e um conjunto de vetores que irá armazenar em breve os posicionamentos dos sensores:

```
Vector4 posicaoIniSensor;
Vector4 posicaoFinSensor;

Vector4[] conjPosicaoIniSensor = new Vector4[conteudoSensores["sensores"].AsArray.Count];
Vector4[] conjPosicaoFinSensor = new Vector4[conteudoSensores["sensores"].AsArray.Count];
```

Como informado anteriormente, utilizamos o "SimpleJSON" para manipular o conteúdo do arquivo JSON de modo mais fácil, principalmente ao obter as informações em array. Enquanto houver sensores declarados, um objeto é incluído no atributo da classe Aviao e posteriormente, são definidos os atributos do sensor de acordo com as informações vindas do arquivo "sensores.json". O atributo temperaturas do objeto Sensor, é definido no momento em que a leitura do arquivo "temperaturas.json". A cada volta, é criado um objeto de Temperatura e seus atributos são definidos de acordo com as informações vindas deste arquivo.

```
for(int c = 0; c < conteudoSensores["sensores"].AsArray.Count; c++) {
        Sensores[c] = new Sensor();

        Sensores[c].id = conteudoSensores["sensores"][c]["id"].AsInt;
        Sensores[c].desc = conteudoSensores["sensores"][c]["desc"];
        Sensores[c].pos_ini_x = conteudoSensores["sensores"][c]["pos_ini_x"].AsFloat;
        Sensores[c].pos_ini_y = conteudoSensores["sensores"][c]["pos_ini_y"].AsFloat;
        Sensores[c].pos_fin_x = conteudoSensores["sensores"][c]["pos_fin_x"].AsFloat;
        Sensores[c].pos_fin_y = conteudoSensores["sensores"][c]["pos_fin_y"].AsFloat;
        Sensores[c].temperatura = new Temperatura[conteudoTemperaturas["temperaturas"].AsArray.Count];

        for(int d = 0; d < conteudoTemperaturas["temperaturas"].AsArray.Count; d++) {
                if(Sensores[c].id == conteudoTemperaturas["temperaturas"][d]["id_sensor"].AsInt) {
                        Sensores[c].temperatura [d] = new Temperatura ();

                        Sensores[c].temperatura[d].data =
conteudoTemperaturas["temperaturas"][d]["data_hora"].ToString().Substring(1,10);
                        Sensores[c].temperatura[d].hora =
conteudoTemperaturas["temperaturas"][d]["data_hora"].ToString().Substring(12,8);
                        Sensores[c].temperatura[d].temp_normal =
conteudoTemperaturas["temperaturas"][d]["temp_normal"].AsFloat;
                        Sensores[c].temperatura[d].temperatura =
conteudoTemperaturas["temperaturas"][d]["temperatura"].AsFloat;
                }
        }

        posicaoIniSensor = new Vector4(Sensores[c].pos_ini_x, Sensores[c].pos_ini_y, 0, 0);
        posicaoFinSensor = new Vector4(Sensores[c].pos_fin_x, Sensores[c].pos_fin_y, 0, 0);

        conjPosicaoIniSensor[c] = posicaoIniSensor;
        conjPosicaoFinSensor[c] = posicaoFinSensor;
```

```
}
```

No final de cada volta do primeiro loop, vemos que a posição é armazenada no conjunto de posições. Isto se deve ao fato de que o conteúdo deste conjunto será transmitido ao Shader vinculado ao modelo 3D, que é o “TCC/Sensor”. A atribuição do conjunto no Shader é realizada através da função “SetVectorArray”, assim como é definida também a quantidade de sensores através da função “SetInt”. Conforme trecho abaixo podemos ver que para acessar o Shader, é necessário acessar o componente “MeshRenderer” do modelo 3D e posteriormente acessar o material vinculado:

```
this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetVectorArray("_PosicoesIniciais",
conjPosicaoIniSensor);

this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetVectorArray("_PosicoesFinais",
conjPosicaoFinSensor);

this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetInt("_qSensores", Sensores.Length);
```

Após a execução do inicializador, o próximo método é o “Update”, que é executado constantemente. Dentro deste método, foi definido um conjunto de teclas que, ao serem pressionadas “rotacionam” o objeto 3D, permitindo melhor visualizar os sensores espalhados durante a execução da cena:

```
void Update () {
        if(Input.GetKey("up") || Input.GetKey("w")){
                transform.Rotate(Vector3.left * 2, Space.World);
        }
        else if(Input.GetKey("down") || Input.GetKey("s")){
                transform.Rotate(Vector3.right * 2, Space.World);
        }
        else if(Input.GetKey("left") || Input.GetKey("a")){
                transform.Rotate(Vector3.up * 2);
        }
```

```
        else if(Input.GetKey("right") || Input.GetKey("d")){
                transform.Rotate(Vector3.down * 2);
        }
```

Posteriormente, é declarado um vetor e um conjunto de vetores que depois armazenará a temperatura esperada do sensor e mais um vetor e um conjunto de vetores que depois armazenará a temperatura do sensor naquele instante. São declaradas duas variáveis, onde cada uma armazena a data e hora atual, respectivamente:

```
Vector4 tempNormalSensor= new Vector4(0, 0, 0, 0);
Vector4 temperaturaSensor = new Vector4(0, 0, 0, 0);
Vector4[]conjTempNormalSensor = new Vector4[Sensores.Length];
Vector4[] conjTemperaturaSensor = new Vector4[Sensores.Length];

string data = System.DateTime.Now.ToString("yyyy-MM-dd");
string hora = System.DateTime.Now.ToString("HH:mm:ss");
```

Logo após a definição, um loop é executado, enquanto houver sensores. A cada volta, é executado um segundo loop que será executado enquanto aquele sensor houver temperaturas vinculadas. Em seguida uma validação é feita verificando se a data atual é a mesma que a informação das temperaturas, a fim de rejeitar informações de dias anteriores. Por fim, é realizado outro teste, verificando se a hora atual (considerando apenas as horas e os minutos) é a mesma que a informação das temperaturas, a fim de apresentar somente as informações do instante atual. Caso o resultado do teste seja “verdadeiro”, o conjunto de temperaturas declarado no inicio do método “Update” receberá o conjunto de temperaturas de todos os sensores daquele instante e posteriormente, atribuirá seu conteúdo na variável do Shader “TCC/Sensor”, vinculada no Material do componente “MeshRenderer” do modelo 3D em questão.

```
for(int c = 0; c < Sensores.Length; c++) {
        for(int d = 0; d < Sensores[c].temperatura.Length; d++) {
```

```
                    if(Sensores[c].temperatura[d] != null){
                        if(data == Sensores[c].temperatura[d].data) {
                            if(hora.Substring(0, 5) == Sensores[c].temperatura[d].hora.Substring(0, 5)) {
                                tempNormalSensor  = new
Vector4(Sensores[c].temperatura[d].temp_normal,0,0,0);
                                temperaturaSensor = new
Vector4(Sensores[c].temperatura[d].temperatura,0,0,0);


                                conjTempNormalSensor [c] = tempNormalSensor ;
                                conjTemperaturaSensor[c] = temperaturaSensor;
                            }
                        }
                    }
            }
}


this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetVectorArray("_TempNormal",
conjTempNormalSensor);
this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetVectorArray("_Temperaturas",
conjTemperaturaSensor);
```

Realizando a atribuição do conjunto de temperaturas no final do método “Update”, o Shader fará com que o efeito de Mapa de Calor seja atualizado constantemente, pois, conforme informado anteriormente, este método é executado a cada frame.

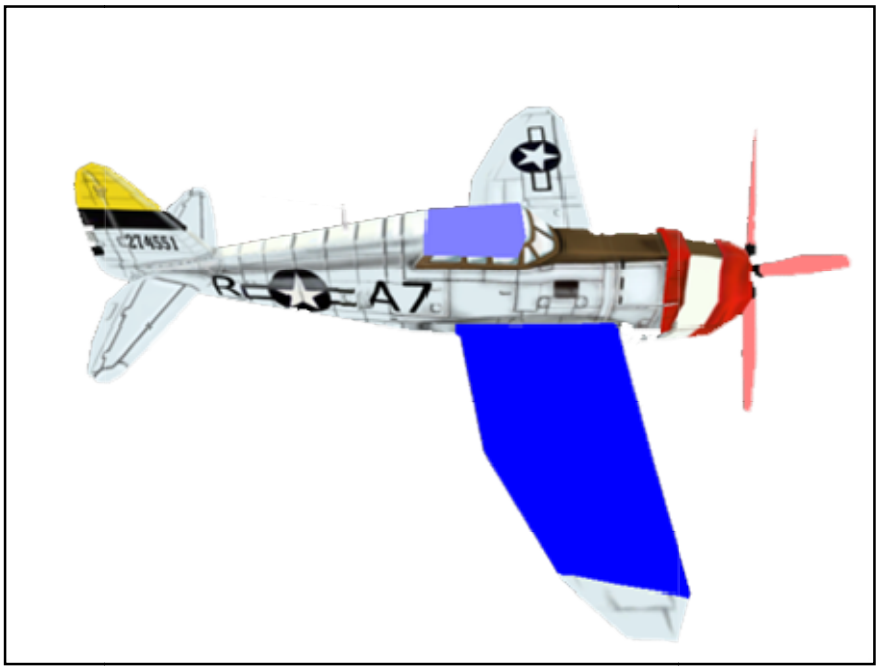
CAPÍTULO 4 – ESTUDO DE CASO

Após a implementação da solução, foi necessário realizar diversos testes de aplicação, a fim de validar o funcionamento do trabalho.

4.1 – Aplicação dos testes com a solução proposta

Na figura abaixo, é possível visualizar o resultado final do projeto, exibindo que em um determinado instante, a “Parte Superior da Asa Direita” está com uma sensação térmica “Muito Fria”, enquanto o sensor localizado na região da “Cabine” apresenta a sensação térmica considerada “Fria” e por fim, a “Hélice” está a ter a sensação térmica considerada “Quente”:

Figura 14 – Resultado Final



Fonte: Elaborado pelo Autor

Pensando-nos mais variados testes a serem realizados, foi elaborada uma planilha que através de colunas como: Código, Descrição, Posição Inicial X, Posição Inicial Y, Posição Final X e Posição Final Y; seja possível facilitar a geração da declaração daquele sensor no

arquivo JSON “sensores.json”. Para isto, foi utilizada a função “CONCATENAR” do Microsoft Office Excel, que permite montar um texto com valores das células junto a textos pré-definidos. Com isto, a declaração de sensores ficou muito mais fácil de ser realizada para fins de testes.

Tabela 2 – Declaração dos Sensores para formato JSON (Excel)

Id	**Desc**	**Pos. Ini. X**	**Pos. Ini. Y**	**Pos. Fin. X**	**Pos. Fin. Y**	**Concatenação**
0	Adesivo da Helice	0,60	0,30	0,70	0,80	{ "id":"0", "desc":"Adesivo da Helice", "pos_ini_x":"0.6", "pos_ini_y":"0.3", "pos_fin_x":"0.7", "pos_fin_y":"0.8" },
1	Cabine	0,87	0,57	0,98	0,68	{ "id":"1", "desc":"Cabine", "pos_ini_x":"0.87", "pos_ini_y":"0.57", "pos_fin_x":"0.98", "pos_fin_y":"0.68" },

Além de facilitar a declaração de sensores, foi elaborada outra planilha que facilita também a declaração de elementos no arquivo JSON, porém agora é para o arquivo “temperaturas.json”. Utilizando o mesmo conceito de concatenação entre valores de colunas, foi possível declarar e conferir o resultado final na Unity com a planilha. Porém esta planilha não foi utilizada apenas facilitar na declaração das temperaturas, mas também como conferência do resultado esperado. Utilizando a “Formatação Condicional” do Microsoft Office Excel, foi possível definir que de acordo com o valor do número, a célula teria sua cor de fundo alterada para a mesma cor estabelecida no Shader, a fim uma melhor identificação e validação.

Tabela 3 – Declaração para formato JSON e Validação das Temperaturas (Excel)

Id	**Data**	**Hora**	**Temp Normal**	**Temperatura**	**Linha do Arquivo "temperaturas.json"**
0	2016-10-14	20:10	13,00	13,00	{"id_sensor":"0", "data_hora":"2016-10-14 20:10:00", "temp_normal ":"13", "temperatura":"13" },
0	2016-10-14	20:11	13,00	19,00	{"id_sensor":"0", "data_hora":"2016-10-14 20:11:00", "temp_normal ":"13", "temperatura":"19" },
1	2016-10-14	20:10	25,00	35,00	{"id_sensor":"1", "data_hora":"2016-10-14 20:10:00", "temperatura":"25", "temperatura":"35" },

2	2016-10-14	20:10	5,00	-1,00	{"id_sensor":"2", "data_hora":"2016-10-14 20:10:00", "temp_normal":"5", "temperatura":"-1" },

É importante destacar que estas planilhas não fazem parte do projeto. Elas apenas auxiliaram no momento de declarar os sensores e as temperaturas no formato JSON e principalmente, na validação dos testes.

4.2 – Complemento da Solução Proposta

O projeto tem como proposto aplicar um efeito de mapa de calor sobre um modelo 3D. Para auxiliar no melhor entendimento das informações apresentadas, um complemento foi elaborado, onde ao invés de apresentar apenas um modelo 3D com a temperatura atual de cada sensor, são apresentados mais dois modelos: um que exiba a temperatura esperada (considerada normal para cada sensor) e outro que apresente a diferença entre as temperaturas de cada sensor.

Para cada novo modelo criado, foi necessário criar um script Shader semelhante ao principal modelo deste projeto, porém com a edição do Fragment Shader. No modelo que apresenta a temperatura esperada, foi alterada a variavel em que o teste era aplicado. Abaixo segue um exemplo de que os testes que eram aplicados na variável "_Temperaturas[c]" foram alterados para "_TempNormal[c]", que recebe o valor da temperatura esperada no Shader deste modelo. É importante considerar que foi necessário também criar um novo material para estes modelos, visto que o Shader é vinculado ao Material e não ao modelo 3D:

```
if(_TempNormal[c].x < 0){
        // .............
        if(_TempNormal[c].x < 15){
                // .............
                if(_TempNormal[c].x < 30){
                        // .............
                        if(_TempNormal[c].x >= 30){
                                // .............
```

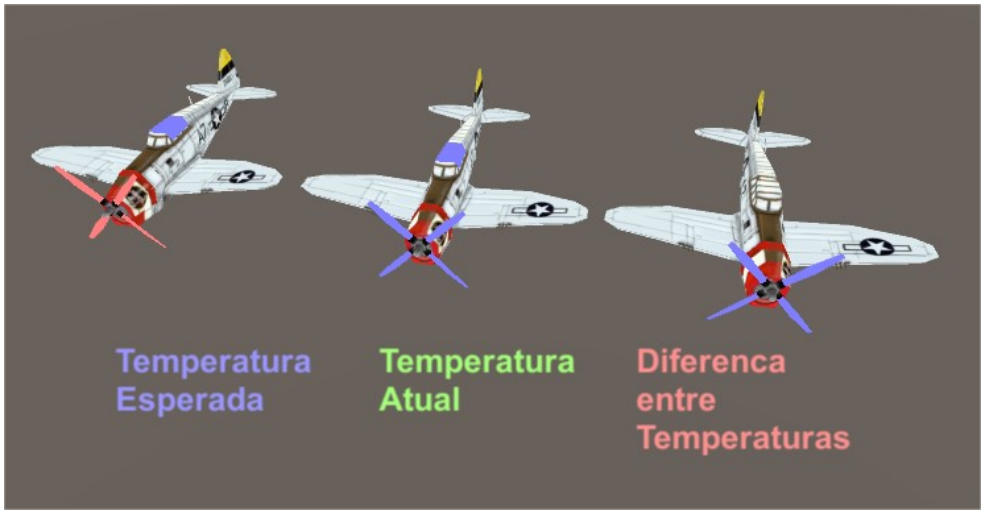
Por fim, para o último modelo, foram ignorados todos os testes de temperaturas do anterior, sendo feito um teste que verifica se o valor da temperatura atual é maior que a

temperatura normal. Se for, aquele sensor terá a cor vermelha clara sobreposta. Se for menor, aquele sensor terá a cor azul clara sobreposta. Caso não retorne verdadeiro em nenhum dos dois testes, será considerada a cor padrão:

```
if(_Temperaturas[c].x > _TempNormal[c].x){
        texturaSaida = _corTempQuente;
}
else{
        if(_Temperaturas[c].x < _TempNormal[c].x){
                texturaSaida = _corTempFrio;
        }
}
```

Com isto, se o terceiro modelo apresentar a cor vermelha clara na região do sensor, é entendido que a temperatura atual daquele sensor está mais elevada do que o esperado. Assim como se for apresentada a cor azul clara, significará que a temperatura atual daquele sensor está abaixo do que o esperado. Abaixo uma figura que apresentam os três modelos, um ao lado do outro:

Figura 15 – Resultado Final (Complemento)



Fonte: Elaborado pelo Autor

Considerando o exemplo acima, é possível entender que a temperatura normal da hélice deve ser uma temperatura considerada quente, mas naquele instante a temperatura era considerada baixa, resultando assim em um esfriamento daquela parte/região. A temperatura normal da cabine deve estar fria e naquele instante a temperatura ideal era a mesma que a temperatura atual, assim a região daquele sensor no modelo 3D não teve suas cores alteradas, visto que ele está conforme esperado.

Com esta solução complementar, mas especificamente no terceiro modelo, é possível apresentar ao usuário somente as regiões cuja temperatura naquele instante podem vir a preocupar, pois dão indícios de falhas e/ou problemas.

CAPÍTULO 5 – CONCLUSÃO

Após a implementação da solução e aplicação dos devidos testes, os objetivos propostos no início do trabalho são concluídos com êxito: a representação da temperatura exata de um determinado objeto (do mundo real) naquele determinado instante em um modelo 3D correspondente; a criação de um método de mapeamento dos sensores em suas determinadas regiões no modelo 3D, representando fielmente o objeto do mundo real; e a geração do efeito de mapa de calor sobre o modelo 3D, permitindo identificar de forma rápida uma possível falha e/ou problema.

O resultado final deste projeto é satisfatório. A solução pode ser aplicada em qualquer objeto, como por exemplo, em um "carro", uma "máquina industrial", etc., independente da maneira como a informação é recebida, seja ela através de arquivos JSON, através da Internet das Coisas ou mesmo utilizando informações pertencentes a caixa preta de um avião.

O projeto permite visualizar a temperatura de partes/peças do equipamento em seu modelo 3D correspondente e auxilia também durante o processo da manutenção preditiva, onde as empresas possam isolar os "objetos reais", aplicarem testes de stress e obterem informações em tempo real e o melhor de tudo, permitindo o rápido entendimento graças à representação visual do modelo 3D e sua temperatura. Tudo isto aplicando a Teoria do Ajuste Cognitivo de Iris Vessey, levando o conceito de que informações apresentadas de modo gráfico/visual são muitas vezes melhor entendidas e bem interpretadas pelos seus utilizadores.

5.1 – Limitações e trabalhos futuros

O projeto possui limitações: entre os sensores está sendo apresentada a cor da textura, ao invés de apresentar uma cor degrade entre o sensor da Cabine, o sensor da Parte Superior da Asa Direita e o sensor da Hélice. Como a sobreposição das cores dos sensores se dá pelo Shader, a interpolação se torna possível, porém é necessário considerar que os modelos 3D são constituídos de diversas malhas (Meshes) e com isto a interpolação é dificultada, porém não é impossível de se realizar.

Além das limitações descritas acima, futuros pesquisadores podem dar seqüência neste, oferecendo uma maneira automática/melhor para declarar o posicionamento dos

sensores, substituindo a planilha de Excel criada para esta finalidade.

Outra sugestão de aperfeiçoamento seria incluir na estrutura JSON dois atributos, sendo um limite inferior e um limite superior de temperatura daquele sensor. Permitindo assim, a solução verificar se aquela temperatura esta dentro do esperado.

Por fim, outra possível sugestão de aperfeiçoamento da solução proposta é permitir que os sensores sejam declarados em formas livres, ao contrário do retângulo que é gerado atualmente. Ou seja, permitir que a representação de um sensor seja feita através de um círculo, um triângulo, ou, melhor ainda, por um polígono fechado, côncavo e com um número infinito de vértices.

REFERÊNCIAS BIBLIOGRÁFICAS

AKENINE-MÖLLER, Tomas; HAINES, Eric; HOFFMAN, Naty. **Real-time rendering**. CRC Press, 2008.

AZEVEDO, Eduardo, e Aura Conci. **Computação gráfica: teoria e prática.** Elsevier, 2003.

BAILEY, Mike; CUNNINGHAM, Steve. **Graphics shaders: theory and practice.** CRC Press, 2016.

EVANGELISTA, Bruno et al. **Renderização de cenas tridimensionais nao-fotorealistas explorando hardware programável**. 2010.

FEIJÓ, Bruno, P. Pagliosa, and E. Clua. **Visualização, simulação e games.** Atualizações em Informática, K. Breitman and R. Anido, eds., Editora PUC-Rio (2006).

FOGGIATO, J. A., Neri Volpato, and A. C. Bontorin. **Recomendações para Modelagem em Sistemas CAD-3D.** 4º Congresso Brasileiro de Engenharia de Fabricação COBEF, São Pedro. Vol. 15. 2008.

FONSECA, Rúben; SIMOES, Alberto. **Alternativas ao XML: YAML e JSON**. 2007.

LIMA, Alex de Souza Campelo. **Aproximação experimental da complexidade assintótica de shaders para dispositivos móveis utilizando OpenGL** ES. 2014.

PASSOS, Erick Baptista, et al. **Tutorial: Desenvolvimento de jogos com unity 3d.** VIII Brazilian Symposium on Games and Digital Entertainment. 2009.

TEIXEIRA, Fernando A. et al. **SIoT–Defendendo a Internet das Coisas contra Exploits.** Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), 2014.

UNITY, **Manual: Writing Shaders**. Disponível em <https://docs.unity3d.com/Manual/ShadersOverview.html> Acesso em 01 de Novembro de 2016.

VESSEY, I. **Cognitive Fit: a theory-based analysis of the graphs versus tables literature.** Decision Science, v. 22, n. 2, p. 219-240, 1991.

ZUCCONI, Alan; LAMMERS, Kenneth. **Unity 5. x Shaders and Effects Cookbook.** 2016.

APÊNDICE A – CÓDIGO SHADER

```
Shader "TCC/Sensor" {
	Properties {
		_corTempFrio("Preenchimento Frio", Color) = (0.50,0.50,1.00,1)
		_corTempMuitoFrio("Preenchimento Muito Frio", Color) = (0.00,0.00,1.00,1)
		_corTempMuitoQuente("Preenchimento Muito Quente", Color) = (1.00,0.00,0.00,1)
		_corTempQuente("Preenchimento Quente", Color) = (1.00,0.50,0.50,1)

		_texturaPadrao("Textura", 2D) = "Modelos/DownNecker/done2.png" {}
	}

	SubShader {
		Tags { "RenderType"="Opaque" }
		LOD 100

		Pass {
			CGPROGRAM

			#pragma vertex funcVertices
			#pragma fragment funcFragmentos

			#include "UnityCG.cginc"

			struct EstruturaDados {
				float2 coordenada : TEXCOORD0;
				float4 posicao : SV_POSITION;
				fixed4 cor : COLOR;
			};

			sampler2D _texturaPadrao;

			float4 _texturaPadrao_ST;

			fixed4 _corTempFrio;
			fixed4 _corTempMuitoFrio;
			fixed4 _corTempMuitoQuente;
			fixed4 _corTempQuente;

			int _qSensores = 0;
```

```
                float2 _PosicoesIniciais[100];
                float2 _PosicoesFinais[100];
                float2 _TempNormal[100];
                float2 _Temperaturas[100];

                EstruturaDados funcVertices (float4 pPosicao : POSITION, float4 pCoordenada :
TEXCOORD0, float4 pCor : COLOR) {
                        EstruturaDados estrutura;

                        estrutura.posicao = mul(UNITY_MATRIX_MVP, pPosicao);
                        estrutura.coordenada = TRANSFORM_TEX(pCoordenada, _texturaPadrao);
                        estrutura.cor = pCor;

                        return estrutura;
                }

                fixed4 funcFragmentos (EstruturaDados estrutura) : SV_Target {
                        fixed4 texturaSaida = tex2D(_texturaPadrao,estrutura.coordenada) * estrutura.cor;

                        for(int c = 0; c < _qSensores; c++){
                                if(estrutura.coordenada.x >= _PosicoesIniciais[c].x &&
estrutura.coordenada.x <= _PosicoesFinais[c].x && estrutura.coordenada.y >= _PosicoesIniciais[c].y &&
estrutura.coordenada.y <= _PosicoesFinais[c].y){
                                        if(_Temperaturas[c].x < 0){
                                                texturaSaida = _corTempMuitoFrio;
                                        }
                                        else{
                                                if(_Temperaturas[c].x < 15){
                                                        texturaSaida = _corTempFrio;
                                                }
                                                else{
                                                        if(_Temperaturas[c].x < 30){
                                                                texturaSaida = _corTempQuente;
                                                        }
                                                        else{
                                                                if(_Temperaturas[c].x >= 30){
                                                                        texturaSaida =
_corTempMuitoQuente;
                                                                }
                                                        }
                                                }
```

```
                                                }
                                            }
                                        }

                                        return texturaSaida;
                                    }

                                    ENDCG
                                }
            }
}
```

APÊNDICE B – CÓDIGO C#

```
using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.IO;
using SimpleJSON;

public class Aviao : MonoBehaviour {

	Sensor[] Sensores;

	string CarregarConteudoArquivo(string nomearquivo) {
		string linhas = "";
		string retorno = "";

		StreamReader leitura = new StreamReader(nomearquivo, Encoding.Default);

		using(leitura) {
			do {
				linhas = leitura.ReadLine();

				if(linhas != null) {
					retorno += linhas;
				}
			} while(linhas != null);

			leitura.Close();

			return retorno;
		}
	}

	Color DefinirTemperatura(float temp){
		Color corSensor = new Color (0.50F, 0.50F, 0.50F, 1.00F);
		Color corFria = new Color (0.50F, 0.50F, 1.00F, 1.00F);
		Color corFriaDemais = new Color (0.00F, 0.00F, 1.00F, 1.00F);
		Color corQuente = new Color (1.00F, 0.50F, 0.50F, 1.00F);
```

```
		Color corQuenteDemais = new Color (1.00F, 0.00F, 0.00F, 1.00F);

		if(temp < 0) {
			corSensor = corFriaDemais;
		} else if(temp < 15) {
			corSensor = corFria;
		} else if(temp < 30) {
			corSensor = corQuente;
		} else {
			corSensor = corQuenteDemais;
		}

		return corSensor;
	}

	public class Temperatura {
		public string data;
		public string hora;
		public float temp_normal;
		public float temperatura;
	}

	public class Sensor {
		public int id;
		public string desc;
		public float pos_ini_x;
		public float pos_ini_y;
		public float pos_fin_x;
		public float pos_fin_y;
		public Temperatura[] temperatura;
	}

	void apresentarDadosSensor(Sensor sensor) {
		Debug.Log("--- Dados do Sensor ---");
		Debug.Log("Id: "+sensor.id);
		Debug.Log("Descricao: "+sensor.desc);
		Debug.Log("Posicionamento Inicial do Eixo X: "+sensor.pos_ini_x);
		Debug.Log("Posicionamento Inicial do Eixo Y: "+sensor.pos_ini_y);
		Debug.Log("Posicionamento Final do Eixo X: "+sensor.pos_fin_x);
		Debug.Log("Posicionamento Final do Eixo Y: "+sensor.pos_fin_y);
	}
```

```
	void apresentarDadosTemperatura(Temperatura temperatura) {
		Debug.Log("--- Dados da Temperatura ---");
		Debug.Log("Data: "+temperatura.data);
		Debug.Log("Hora: "+temperatura.hora);
		Debug.Log("Temperatura Normal: "+temperatura.temp_normal+"° C");
		Debug.Log("Temperatura: "+temperatura.temperatura+"° C");
	}

	void Start() {
		string leituraSensores = "";
		string leituraTemperaturas = "";

		leituraSensores = CarregarConteudoArquivo("Assets/_Leitura/sensores.json");
		leituraTemperaturas = CarregarConteudoArquivo("Assets/_Leitura/temperaturas.json");

		var conteudoSensores = JSON.Parse(leituraSensores);
		var conteudoTemperaturas = JSON.Parse(leituraTemperaturas);

		Sensores = new Sensor[conteudoSensores["sensores"].AsArray.Count];

		Vector4 posicaoIniSensor;
		Vector4 posicaoFinSensor;

		Vector4[] conjPosicaoIniSensor = new Vector4[conteudoSensores["sensores"].AsArray.Count];
		Vector4[] conjPosicaoFinSensor = new Vector4[conteudoSensores["sensores"].AsArray.Count];

		for(int c = 0; c < conteudoSensores["sensores"].AsArray.Count; c++) {
			Sensores[c] = new Sensor();

			Sensores[c].id = conteudoSensores["sensores"][c]["id"].AsInt;
			Sensores[c].desc = conteudoSensores["sensores"][c]["desc"];
			Sensores[c].pos_ini_x = conteudoSensores["sensores"][c]["pos_ini_x"].AsFloat;
			Sensores[c].pos_ini_y = conteudoSensores["sensores"][c]["pos_ini_y"].AsFloat;
			Sensores[c].pos_fin_x = conteudoSensores["sensores"][c]["pos_fin_x"].AsFloat;
			Sensores[c].pos_fin_y = conteudoSensores["sensores"][c]["pos_fin_y"].AsFloat;
			Sensores[c].temperatura = new
Temperatura[conteudoTemperaturas["temperaturas"].AsArray.Count];

			for(int d = 0; d < conteudoTemperaturas["temperaturas"].AsArray.Count; d++) {
				if(Sensores[c].id == conteudoTemperaturas["temperaturas"][d]["id_sensor"].AsInt) {
```

```
                                        Sensores[c].temperatura [d] = new Temperatura ();


                                        Sensores[c].temperatura[d].data          =
conteudoTemperaturas["temperaturas"][d]["data_hora"].ToString().Substring(1,10);
                                        Sensores[c].temperatura[d].hora =
conteudoTemperaturas["temperaturas"][d]["data_hora"].ToString().Substring(12,8);
                                        Sensores[c].temperatura[d].temp_normal =
conteudoTemperaturas["temperaturas"][d]["temp_normal"].AsFloat;
                                        Sensores[c].temperatura[d].temperatura =
conteudoTemperaturas["temperaturas"][d]["temperatura"].AsFloat;
                                    }
                                }


                                posicaoIniSensor = new Vector4(Sensores[c].pos_ini_x, Sensores[c].pos_ini_y, 0, 0);
                                posicaoFinSensor = new Vector4(Sensores[c].pos_fin_x, Sensores[c].pos_fin_y, 0, 0);


                                conjPosicaoIniSensor[c] = posicaoIniSensor;
                                conjPosicaoFinSensor[c] = posicaoFinSensor;
                        }


        this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetVectorArray("_PosicoesIniciais",
conjPosicaoIniSensor);

        this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetVectorArray("_PosicoesFinais",
conjPosicaoFinSensor);


                this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetInt("_qSensores",
Sensores.Length);
        }


        void Update () {
                if(Input.GetKey("up") || Input.GetKey("w")){
                        transform.Rotate(Vector3.left * 2, Space.World);
                }
                else if(Input.GetKey("down") || Input.GetKey("s")){
                        transform.Rotate(Vector3.right * 2, Space.World);
                }
                else if(Input.GetKey("left") || Input.GetKey("a")){
                        transform.Rotate(Vector3.up * 2);
                }
                else if(Input.GetKey("right") || Input.GetKey("d")){
                        transform.Rotate(Vector3.down * 2);
                }
```

```
                Vector4 temperaturaSensor = new Vector4(0, 0, 0, 0);
                Vector4 tempNormalSensor = new Vector4(0, 0, 0, 0);

                Vector4[] conjTemperaturaSensor = new Vector4[Sensores.Length];
                Vector4[]conjTempNormalSensor = new Vector4[Sensores.Length];

                string data = System.DateTime.Now.ToString("yyyy-MM-dd");
                string hora = System.DateTime.Now.ToString("HH:mm:ss");

                for(int c = 0; c < Sensores.Length; c++) {
                        for(int d = 0; d < Sensores[c].temperatura.Length; d++) {
                                if(Sensores[c].temperatura[d] != null){
                                        if(data == Sensores[c].temperatura[d].data) {
                                                if(hora.Substring(0, 5) ==
Sensores[c].temperatura[d].hora.Substring(0, 5)) {
                                                        tempNormalSensor = new
Vector4(Sensores[c].temperatura[d].temp_normal,0,0,0);
                                                        temperaturaSensor = new
Vector4(Sensores[c].temperatura[d].temperatura,0,0,0);

                                                        conjTempNormalSensor[c] = tempNormalSensor;
                                                        conjTemperaturaSensor[c] = temperaturaSensor;
                                                }
                                        }
                                }
                        }
                }


        this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetVectorArray("_TempNormal",
conjTempNormalSensor);
        this.gameObject.GetComponentInChildren<MeshRenderer>().material.SetVectorArray("_Temperaturas",
conjTemperaturaSensor);
        }
}
```