Universidade Federal de Pernambuco
Centro de Informática

Bacharelado em Ciência da Computação

Desenvolvimento de Ferramenta de Visualização para Aplicação Web de Modelagem e Análise Quantitativa de PET Scanners

Márcia Cristina Tenorio Cavalcante

Trabalho de Graduação

Recife, Dezembro 2018

Universidade Federal de Pernambuco
Centro de Informática

Desenvolvimento de Ferramenta de Visualização para Aplicação Web de Modelagem e Análise Quantitativa de PET Scanners

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação

Aluna: Márcia Cristina Tenorio Cavalcante
Orientador: Silvio Barros Melo

Recife, Dezembro de 2018

“You see me I be work, work, work, work, work, work.”
—RIHANNA

AGRADECIMENTOS

Agradeço à minha família pelo incentivo, em especial a minha mãe por sempre acreditar em mim. Ao meu namorado por sempre estar ao meu lado, e aos meus amigos por estarem presentes sempre que foi necessário. Também sou grata por todas as orientações de Silvio Melo, e pela imensa ajuda de Daniel Bonifácio, assim como toda equipe do WebGUIGATE. Por fim, gostaria de agradecer as divas do Pop, por me darem forças para seguir adiante, em especial Miley Cyrus, Rihanna, Selena Gomez e Ariana Grande, afinal foi ao som de muitos *We Can't Stop* que este trabalho nasceu. *Thank you, next.*

“Every single night and every single day, i'mma do my thing, i'mma do my thing, so don't you worry about me i'll be okay, i'mma do my thing, 'cause i'mma do my thing.”

—MILEY CYRUS

Resumo

Com intuito de se tornar cada vez mais precisa em seus diagnósticos e tratamentos, a medicina faz uso de soluções como a tomografia por emissão de pósitrons (PET) para atingir seus objetivos. A capacidade de simular tais equipamentos impulsiona o desenvolvimento da área e ajuda a entender as relações físicas exploradas pelo equipamento. Este trabalho propõe uma ferramenta que possibilite a visualização de dispositivos PET através da web. Para o desenvolvimento da mesma, foi usado como base o projeto WebGUIGATE, que é uma interface web para o software GATE. Nele o usuário torna-se apto a gerar scripts de simulações de equipamentos PET, compatíveis com o GATE. Por fim, apresenta-se os resultados obtidos através do desenvolvimento, e possíveis pontos de evolução do sistema WebGUIGATE como um todo.

SUMÁRIO

1. INTRODUÇÃO 8

2. FUNDAMENTAÇÃO 9

2.1 Tomografia por Emissão de Positrões 9
2.2 GATE 11
2.2.1 Geometria 11
2.2.2 Geometria do Scanner 11
2.3 WebGUIGATE 12
2.4 Linguagem para Modelagem de Realidade Virtual 14
2.5 JavaScript 14
2.6 Three.js 14
2.7 JQuery 15
2.8 CSS 15
2.9 HTML 16
2.10 Django 16
2.11 Git 16
2.13 Interface Humano Computador 17

3. METODOLOGIA 18

3.1 WebGUIGATE 18
3.1.1 Arquitetura 18
3.1.2 Scanner 20
3.2 Implementação 22
3.2.1 Back-end 22
3.2.2 Front-end 23

4. RESULTADOS 24

4.1 Visualizador Web de PET Scanners 24

5. CONCLUSÃO 27

REFERÊNCIAS 28

1. INTRODUÇÃO

A medicina está sempre em busca de maneiras de melhorar diagnósticos e tratamentos, tornando-os cada vez mais assertivos. A tomografia por emissão de pósitrons (PET) tem permitido o avanço da medicina nuclear moderna, a partir de aquisições de imagens funcionais, avaliando o metabolismo celular. Amplamente utilizado nos diagnósticos em cardiologia, neurologia e oncologia, equipamentos PET permitem a aquisição de imagens órgãos, e do corpo inteiro (LJUNGBERG; STRAND; KING, 2012).

Fatores físicos como eficiência dos detectores e colimadores, eventos de coincidência não colineares e aleatórios, configuração do scanner, podem ser simulados e para tanto já contamos com alguns *softwares* disponíveis, entre eles, o código aberto GATE (Geant4 Application for Tomographic Emission) (JAN *et al.*, 2004), que é capaz de simular a arquitetura e o comportamento de diversos equipamentos, entre eles o PET *Scanner*. Contudo, esse sistema apresenta alguns desafios, em termos de usabilidade, sobre tudo para usuários não especializados, uma vez que as simulações são geradas a partir de scripts pré-definidos. Também é necessário o uso do ambiente Linux para a execução do *software*.

Afim de vencer alguns desses obstáculos de uso do GATE, foi desenvolvida a aplicação WebGUIGATE (CORDEIRO, [S.d.]). O sistema foi concebido através do *framework* Django, e teve sucesso em seus objetivos, tornando mais amigável para a pessoa usuária, a criação de scripts de simulação GATE.

Se faz notória a contribuição do WebGUIGATE para a comunidade de usuários e desenvolvedores do GATE como um todo, entretanto ainda é possível encontrar algumas oportunidades de melhorias. Atualmente o sistema suporta apenas *scanners* do tipo cilíndrico. Além disso, apesar da tela de configuração de *scanner* conter imagens para auxiliar a pessoa usuária durante a definição dos parâmetros, não existe nenhum *feedback* visual que assegure que o resultado final de fato condiz com o desejado.

Pensando nos pontos de possível evolução do sistema, previamente citados, foi decido criar uma interface na qual o usuário conseguiria ter uma pré-visualização do equipamento *scanner* contido na simulação.

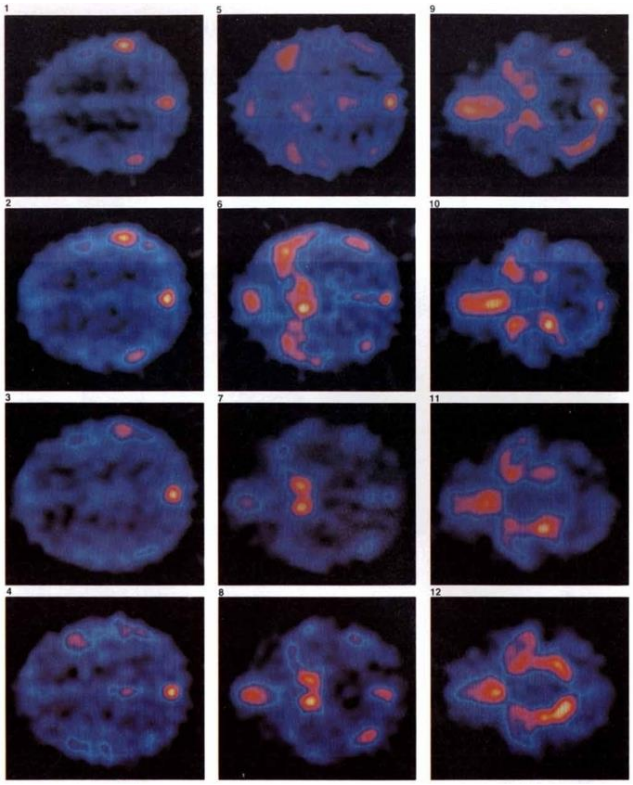
2. FUNDAMENTAÇÃO

Este capítulo apresenta os conceitos mínimos necessários para o total entendimento deste trabalho.

2.1 Tomografia por Emissão de Positrões

A tomografia por emissão de pósitrons (PET), é uma técnica de formação de imagens que se utiliza de moléculas radiomarcadas com radionuclídeos emissores de pósitrons para analisar suas interações nos processos biológicos (BOMBARDA; SOARES JÚNIOR; TERRA FILHO, 2002). Os mesmos se baseiam na detecção de pares de fótons de 511 keV emitidos em direções opostas pela aniquilação do pósitron (β^+) com um elétron vindo do tecido. O resultado desse processo são imagens que revelam a atividade metabólica celular, proporcionando a quantificação da absorção de radionuclídeo na região de interesse (SAHA, 2016). Na Figura 1, podemos ver o exemplo de uma imagem adquirida através de um exame PET.

Figura 1 - Imagens do cérebro humano extraídas de um exame PET.



Fonte: (MICHEL M. TER-POGOSSIAN, 1980)

2.2 GATE

GATE é um sistema computacional de código aberto, voltado a simulações numéricas de sistemas de formação de imagens médicas e radioterapia. A partir do método Monte Carlo, o GATE é capaz de simular a arquitetura e o comportamento de diversos equipamentos, entre eles o PET *Scanner*.

O GATE segue o seguinte algoritmo para realizar uma simulação:

1. Definir a geometria do scanner;
2. Definir a geometria do simulador antropomórfico virtual
3. Estabelecer os processos físicos envolvidos;
4. Iniciar a simulação;
5. Estabelecer o modelo do detector;
6. Estabelecer as fontes;
7. Especificar o formato de saída dos dados;
8. Iniciar a aquisição de dados.

2.2.1 Geometria

A geometria do GATE é baseada em volumes, sendo cada um deles ligados sequencialmente seguindo uma hierarquia. Os mesmos devem conter uma forma, dimensão, posição e material de composição. Por padrão o GATE já contém o volume World que consiste em um paralelepípedo centrado na origem do sistema de coordenadas cartesianas e suas dimensões devem ser definidas pelo usuário. Todos os outros volumes devem ser definidos como “filhos” do volume previamente definido dentro da hierarquia. (OPENGATE, 2017)

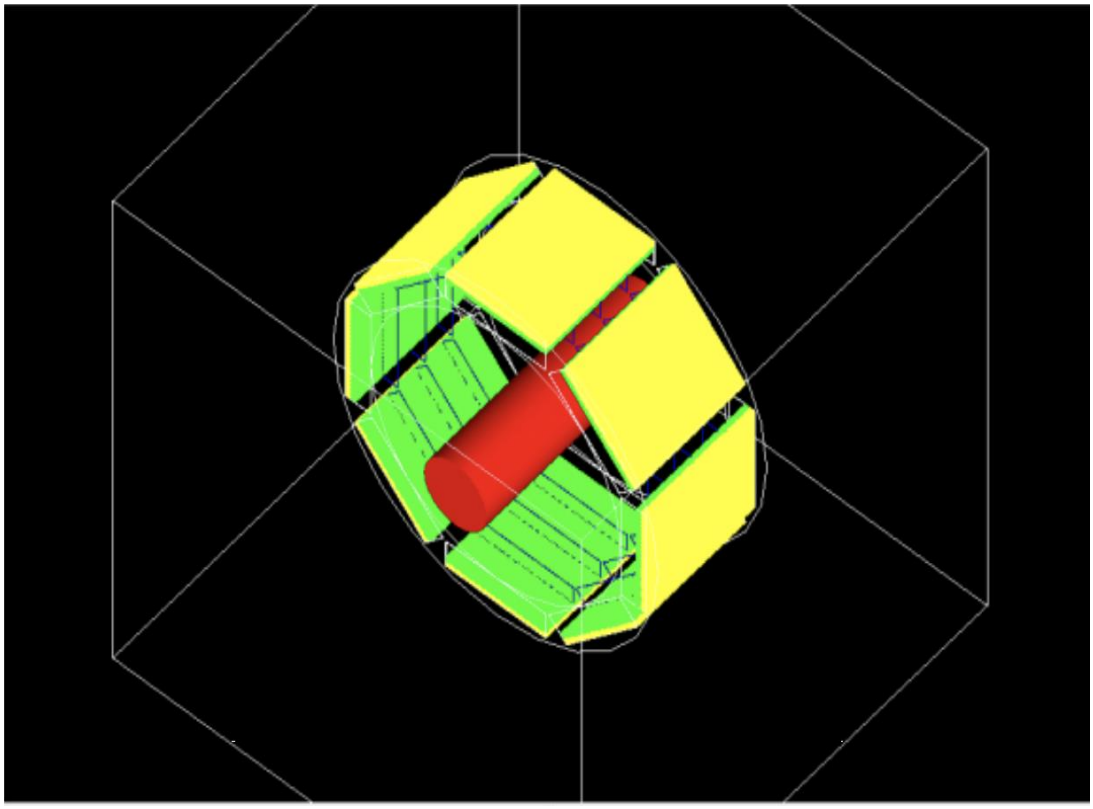
2.2.2 Geometria do Scanner

Antes de ter a geometria do *Scanner* definida, o usuário pode visualizar sua construção interativamente. Para tal, deve-se utilizar uma das opções de interface gráficas disponibilizadas pelo GATE, entre elas pode-se usar a biblioteca gráfica OpenGL oriunda do

GEANT4. O *Scanner* é renderizado como um objeto 3D, onde é possível aplicar transformações em tempo real no mesmo, i.e., rotação e escala. Na Figura 2 é possível ver o exemplo de um PET cilíndrico renderizado no OpenGL.

Ainda é possível exportar a geometria do *Scanner* para que seja visualizada em outras plataformas. Existem diversos formatos disponíveis para exportação, cada um com vantagens e desvantagens, entre eles estão: JPG, PDF e VRML.

Figura 2 - Simulação de um PET cilíndrico no GATE.



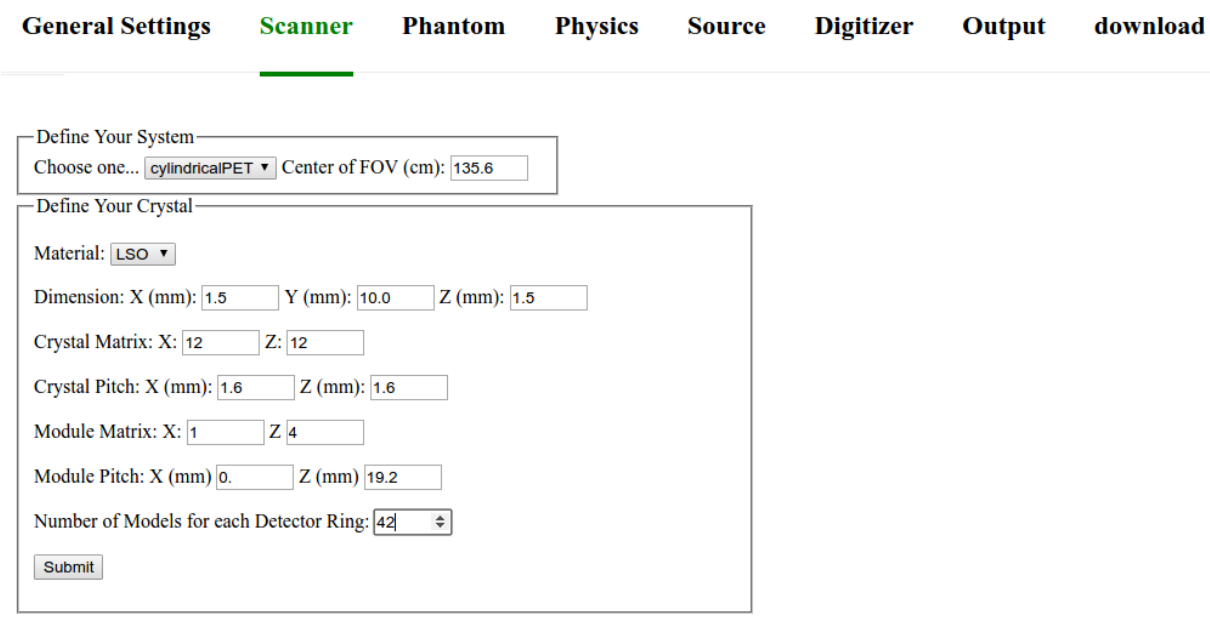
Fonte: (OPENGATE, 2017)

2.3 WebGUIGATE

A instalação do GATE pode torna-se desafiadora, uma vez que requer certo nível de conhecimento de programação de computadores. A fim de superar essa barreira, foi criada a aplicação WebGUIGATE, que nada mais é que uma interface para a invocação de serviços de simulação GATE na *World Wide Web*, através do navegador web. Atualmente a WebGUIGATE suporta simulações de PET e resolve problemas de instalação e codificação GATE, gerando os comandos de macro necessário para realizar uma simulação. (CORDEIRO, [S.d.]).

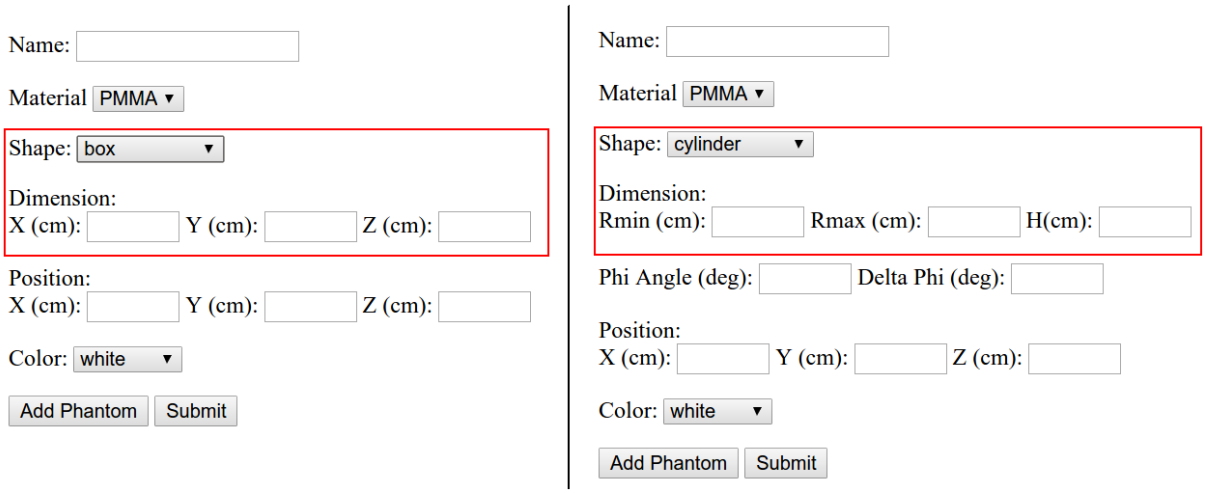
Nas figuras 3 e 4 é exibido o exemplo de algumas telas resultantes desse sistema. Na Figura 3 é notado que o sistema é divido em abas, onde em cada uma das abas a pessoa usuário pode definir as configurações desejadas para sua simulação. Também é importante ressaltar que o sistema é iterativo, i.e., adapta-se às necessidades do usuário em tempo de execução, como é mostrado na Figura 4.

Figura 3 - Página para a modelagem do Scanner.



Fonte: (CORDEIRO, [S.d.])

Figura 4 - Interatividade dos campos em resposta às ações do usuário na aba Phanton.



Fonte: (CORDEIRO, [S.d.])

2.4 Linguagem para Modelagem de Realidade Virtual

A Linguagem para Modelagem de Realidade Virtual (VRML) é um padrão de formato de arquivo para descrever e transmitir via rede objetos 3D e mundos geométricos e de multimídias. Ao criar-se um arquivo VRML, é possível definir diversos atributos, como: vértices, arestas, cor, brilho, transparência, mapas de texturas e etc. Também é possível associar URLs a componentes gráficos, para que usuário seja direcionado para outro endereço ao clicar no então componente (TAUBIN *et al.*, 1998).

Com o intuito de fomentar o desenvolvimento de soluções de realidade virtual na web, VRML tornou-se o formato padrão para descrever cenas 3D nos anos noventa. Seus principais alvos eram aplicações web voltadas à engenharia, visualizações cientificas e entretenimento (TAUBIN *et al.*, 1998). Ainda usado atualmente, é possível encontrar código escrito em JavaScript capaz de renderizar arquivos VRML.

2.5 JavaScript

JavaScript é uma linguagem de alto nível, interpretada, caracterizada por ser dinâmica, fracamente tipada e multiparadigma (W3SCHOOL, [S.d.]). Foi originalmente implementada como parte dos navegadores web para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade de os mesmos acionarem o servidor. Com JavaScript, é possível controlar o navegador, realizando conexões assíncronas e alterar dinamicamente o conteúdo do documento exibido.

Nos dias de hoje é uma das principais linguagens de programação para o lado do cliente em navegadores web. Também é possível encontrá-la em diversos projetos no lado do servidor através de ambientes como node.js. O fato de possuir suporte à programação funcional e fechamento de funções de alta ordem a torna particularmente atrativa para os mais diversos tipos de aplicações.

2.6 Three.js

Exibir uma cena 3D através do navegador, sem dúvida, pode ser desafiador para uma pessoa que esteja iniciando sua jornada no desenvolvimento, especialmente porque além do conhecimento básico sobre programação, também seria necessária a compreensão tanto de

algoritmos de renderização, quanto de formatação de arquivos, para fazer a decodificação do arquivo em uma cena.

Com o intuito de diminuir a complexidade de desenvolvimento desse tipo de aplicação, temos Three.js, que é uma Interface de Programação de Aplicações (API) para JavaScript, empregada na criação e exibição de cenas ou animações 3D no navegador web (THREE.JS, [S.d.]). Esta biblioteca faz uso do WebGL, uma versão web da conhecida biblioteca OpenGL, como base e possibilita a exibição de animações aceleradas em Unidade de Processamento Gráfico (GPU), sem fazer uso de nenhum *plugin* externo, o que torna o sistema mais seguro, independente e fluído. Além de todos os benefícios previamente citados, o código fonte de Three.js está disponível no GitHub para que usuário seja capaz de integrar em sua aplicação.

2.7 JQuery

JQuery é uma biblioteca de código aberto para JavaScript. A sintaxe da mesma foi construída de maneira a simplificar a navegação e a interação com o documento HTML, a seleção de elementos provenientes do Modelo de Documento por Objetos (DOM), criar animações, manipular eventos, desenvolver aplicações de Javascript Assíncrono e XML (AJAX) e a criação de plug-ins. Tais facilidades possibilitam as pessoas desenvolvedoras criarem camadas de abstração para interações de baixo nível de modo simplificado em aplicações web dinâmicas de grande complexidade.

2.8 CSS

Tão importante quanto as funcionalidades de uma aplicação, é como ela se apresenta aos usuários. Dito isso é importante ter alguns mecanismos que tornem possível a configuração de como as informações são expostas na tela do usuário. Para tratar dessa questão fazemos uso da *Cascading Style Sheets* (CSS), que é uma linguagem de folhas de estilo. A mesma é responsável por toda estilização das paginas, i.e., posicionamento dos elementos, estética das fontes, cores, bordas, entre outras características presentes no documento HTML descritos nas paginas web.

2.9 HTML

A Linguagem de Marcacação de Hypertexto (HTML) é a linguagem padrão para criação de paginas e aplicações na web (W3C, 2012). Através das tags HTML é possível estruturar documentos capazes de serem interpretados pelos navegadores disponíveis. Neste trabalho, esta linguagem será usada para criar a estrutura das paginas web que serão posteriormente renderizadas.

2.10 Django

O tempo e a complexidade de desenvolvimento de uma aplicação ou serviço na web, frequentemente aparecem como fator determinístico no momento de escolha da tecnologia a ser usada. Em particular nesse trabalho, ambos aparecem como principais requisitos. Pensando nisso, foi escolhido o *framework* Django para o desenvolvimento.

Django é um *framework* web de alto nível baseado em Python que tem como objetivo acelerar e simplificar o desenvolvimento, uma vez que aposta em um design limpo e bem definido, para que a pessoa desenvolvedora possa focar apenas do desenvolvimento em si.

Características como servidor de desenvolvimento embutido, fácil integração com uma base de dados e baixo acoplamento tornam o desenvolvimento mais intuitivo até para quem não está adaptado aos paradigmas de codificação web. Além de todas as comodidades previamente citadas, Django ainda se destaca por ser uma ferramenta gratuita de código aberto.

2.11 Git

Um grande desafio para projetos que possuem duas ou mais pessoas trabalhando simultaneamente, ou mesmo que está em constante evolução, é manter o rastreamento das mudanças realizadas ao longo do tempo. Para auxiliar no armazenamento e rastreio de alterações no código fonte, foi utilizada a ferramenta Git, que é um sistema de versionamento de arquivos. Com ele é possível paralelizar edições no código fonte, verificar o histórico de edição de um dado arquivo, verificar as pessoas responsáveis por tais mudanças entre outras coisas. Além do que já foi citado, também aumenta a redundância dos dados de código, pois é possível hospedar o código em um repositório na nuvem, e assim tendo um backup seguro

para o código.

2.13 Interface Humano Computador

Durante a implementação de todas as funcionalidades descritas nessa obra, fez-se evidente o seu enfoque em pessoas, uma vez que, são elas as utilizadoras dos serviços. Com base nisso, ficou entendido que seria necessário um estudo intencional sobre como se da a interação entre humanos e computadores.

Interface Humano Computador (IHC) surge como área de pesquisa que tem seu enfoque em fornecer aos pesquisadores e desenvolvedores de sistemas e interativos tanto explicações (o porquê de certo evento estar acontecendo de uma dada forma), como previsões (o que aconteceria se este mesmo evento fosse feito de outra maneira?) para fenômenos de interação usuário-sistema e também resultados práticos para o projeto de interface do usuário (SELBACH SILVEIRA; OLIVEIRA PRATES, 2007).

Diversos elementos são usados para proporcionar uma boa interface em computação, dentre eles pode-se destacar as ferramentas funcionais, capazes de melhorar a segurança, a usabilidade e a utilidades dos sistemas como um todo. As interfaces pensadas em usuários humanos são produtos de trabalhos interdisciplinares, que agregam profissionais de diferentes campos em torno de um objetivo comum de aperfeiçoar a aprendizagem das funcionalidades do sistema.

Nos conceitos básicos de IHC, a denominação de interface amigável ou agradável define a interface capaz de entregar estímulos visuais, como cores, formas, fontes, texturas entre outros elementos, de forma equilibrada e harmônica, visando não saturar a visão nem tampouco sobrecarregar a capacidade de assimilação das pessoas usuários diante do crescente fluxo informacional (BATISTA, 2003). No que diz respeito à interface interativa, refere-se àquela que propicia ao individuo controlar as atividades dos sistemas de computação, de forma mais fácil e ágil, e também de maneiras variadas, sem causar erros durante o uso.

3. METODOLOGIA

Este capítulo tem como objetivo apresentar a ferramenta desenvolvida ao longo deste trabalho, além de demonstrar como a mesma se comunica com as funcionalidades pré-existentes no sistema. Para facilitar o entendimento do trabalho como um todo, vamos dividir a metodologia em duas partes. Sendo a primeira, uma imersão no sistema WebGUIGATE previamente existente e algumas das dificuldades dos usuários. Por fim será descrito o processo de criação da nova funcionalidade.

3.1 WebGUIGATE

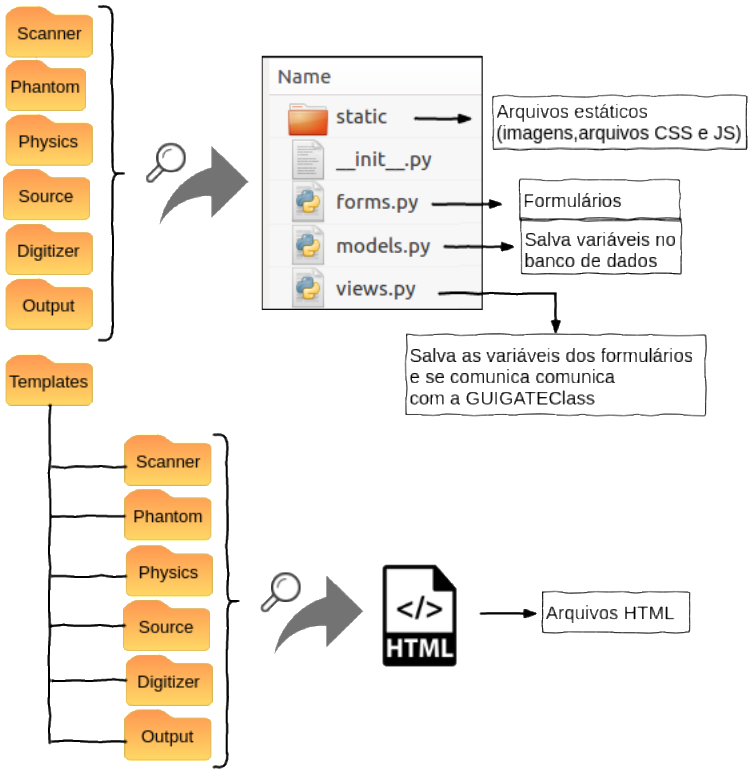
A WebGUIGATE é uma aplicação web desenvolvida em Django, que serve como interface entre o usuário final e o sistema de simulação GATE. Através dela, o usuário é capaz de gerar *scripts* para serem executado no GATE, por consequência diminuindo o esforço inicial necessário para utilizar o *software* de simulação.

3.1.1 Arquitetura

Dentre numerosas vantagens listadas ao usar algum *framework* de desenvolvimento, pode-se destacar a abstração da arquitetura para a ferramenta. Uma vez determinado o mesmo, a pessoa desenvolvedora não precisa mais se preocupar em gerenciar a arquitetura da aplicação, pois o *framework* absorve essa responsabilidade para si.

Aliando a estrutura oferecida pela ferramenta Django, com a linguagem multiplataforma e de sintaxe sucinta de Python foi possível construir a aplicação, sem a necessidade de um estudo mais intencional sobre *design* de *software* ou extensa pesquisa sobre linguagens de desenvolvimento web (HORSTMANN; NECAISE, 2013).

Figura 5 - Esquema de construção do WebGUIGATE.

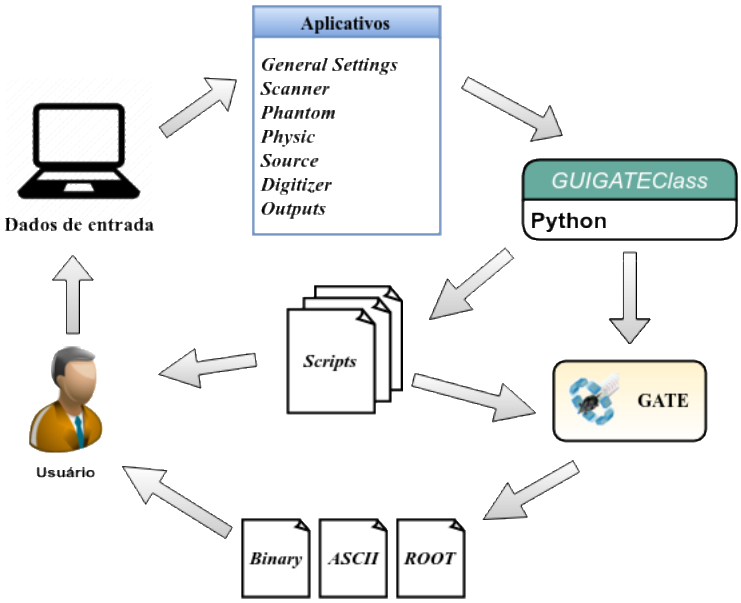


Fonte: (CORDEIRO, [S.d.])

A Figura 5 mostra disposição de arquivos dentro do projeto. Como dito previamente a organização de pastas e documentos é definida pelo *framework*. Para cada sub aplicação, neste caso cada tela, existe uma pasta que contém todos os registros referentes à mesma. Ainda dentro destas pastas, encontram-se outros diretórios, em particular um deles chamado *static*, nele estão contidos todos os arquivos estáticos do sistema, i.e., imagens, folhas de estilo, arquivos JavaScript e etc. Outro importante diretório se chama *template*, onde basicamente estão todos os documentos HTML da aplicação.

Inicialmente o sistema incluía oito telas, sendo sete delas responsáveis por gerar scripts para trechos específicos de uma simulação. Por fim, uma ultima tela encarregada de disponibilizar os arquivos gerados para *download*. Cada sub-aplicação possui formulários para preenchimento de dados de entrada do usuário. Uma vez que os dados são enviados ao servidor, as funções estabelecidas em cada arquivo views.py são acionadas, que por sua vez executarão as classes definidas na classe principal da aplicação (CORDEIRO, [S.d.]). A Figura 6 ilustra a relação do usuário com todo o ecossistema do programa.

Figura 6 - Esquema de funcionamento da WebGUIGATE.



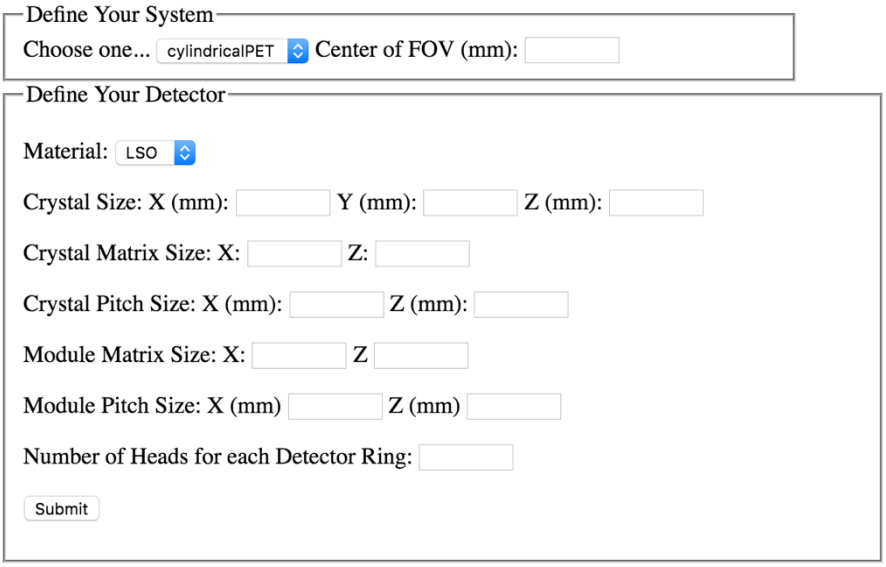
Fonte: (CORDEIRO, [S.d.])

3.1.2 Scanner

Um momento muito importante numa dada simulação, é o de definir a geometria do PET *scanner* a ser usado. No WebGUIGATE, existe uma tela dedicada a esta configuração. Atualmente estão disponíveis no sistema os parâmetros necessários para gerar um *scanner* do tipo cylindricalPET. Em comparação ao manual de uso do GATE, esta tela possui menos parâmetros que o esperado, entretanto esta diferença tem como intenção a simplificação do processo como um todo (CORDEIRO, [S.d.]). A Figura 7 apresenta os parâmetros disponíveis para preenchimento do usuário.

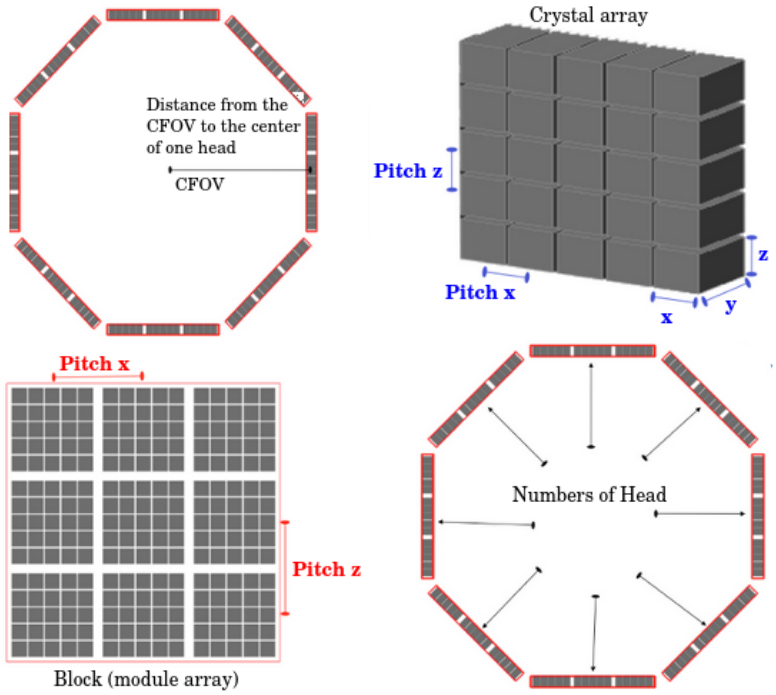
Para auxiliar o usuário na definição desses novos parâmetros, a tela também conta com imagens que ilustram o significado dos mesmos. A Figura 8 mostra as imagens auxiliares exibidas na pagina.

Figura 7 - Tela de configuração do scanner



Fonte: (CORDEIRO, [S.d.])

Figura 8 - Definição dos novos parâmetros para implementação do scanner.



Fonte: (CORDEIRO, [S.d.])

3.2 Implementação

Pode-se dividir a implementação da nova funcionalidade proposta em dois grandes blocos lógicos, blocos estes comumente utilizados na engenharia de *software*, *front-end* e *back-end*. O *front-end* trata-se do lado do cliente, tipicamente da parcela mais visual do sistema, de interação direta com o usuário. Enquanto o *back-end* está relacionado ao lado do servidor, i.e., onde as requisições são processadas e os scripts são gerados de fato.

Afim de manter a rastreabilidade foi usado o sistema de versionamento Git. Através dele, além de manter registradas todas as alterações feitas no código, sendo permitido reverter trechos específicos de código, ainda proporcionou maior autonomia de toda equipe envolvida na construção do WebGUIGATE, uma vez que o Git possibilita a paralelização de código entre os integrantes do time.

A infraestrutura disponível para desenvolvimento e teste da aplicação conta com uma maquina de *kernel* UNIX, processador Intel Core I5 3,1GHz e memória de 8GB.

3.2.1 Back-end

Todo o back-end foi construído para se integrar com o sistema previamente desenvolvido. Sendo assim a linguagem adotada para a criação da nova aplicação foi Python, com o auxilio do *framework* Django para manter a conectividade com o front-end. A máquina incumbida de rodar a aplicação tem como pré-requisitos um *kernel* Unix, assim como a aplicação GATE instalada.

O processo de criação da ferramenta responsável pela pré-visualização do *scanner*, começa ainda na pagina de configuração do mesmo. Ao finalizar o preenchimento de todos os parâmetros necessários para gerar o script de *scanner*, o usuário deve clicar no botão *submit*, a partir daí o servidor processará as entradas para gerar o *script*. Após o gerado o *script*, o sistema emite um comando Unix para sistema operacional do servidor, para que o mesmo execute o script recém-produzido na aplicação GATE. Como saída, o GATE produz um arquivo VRML, que contém a descrição do *scanner* então criado, em formato de objeto. Arquivo este, que posteriormente será usado para renderizar o *scanner* na web.

3.2.2 Front-end

Grande parte do esforço aplicado na construção deste trabalho, está concentrada nessa camada da aplicação. Também nessa camada está contida uma maior variedade de tecnologias. A página base é criada em HTML, este arquivo é responsável tanto por formar um esqueleto básico em que as outras ferramentas sejam capazes de manipular, quanto como estabelecer as dependências necessárias para o projeto.

Para tornar a tela dinâmica, i.e., responsiva a interações do usuário, fez-se uso de algumas bibliotecas disponíveis para JavaScript, entre elas destacam-se JQuery e Three.js. JQuery é usada para criar a ligação entre o código JavaScript e o documento HTML. Já a biblioteca Three.js tem um destaque adicional neste trabalho, uma vez que é através dela que os arquivos VRML são renderizados. O objeto VRLM gerado de antemão pelo back-end, neste ponto é recuperado pelo JavaScript e repassado para a biblioteca, que com o auxilio do WebGL faz o carregamento do *scanner* na tela.

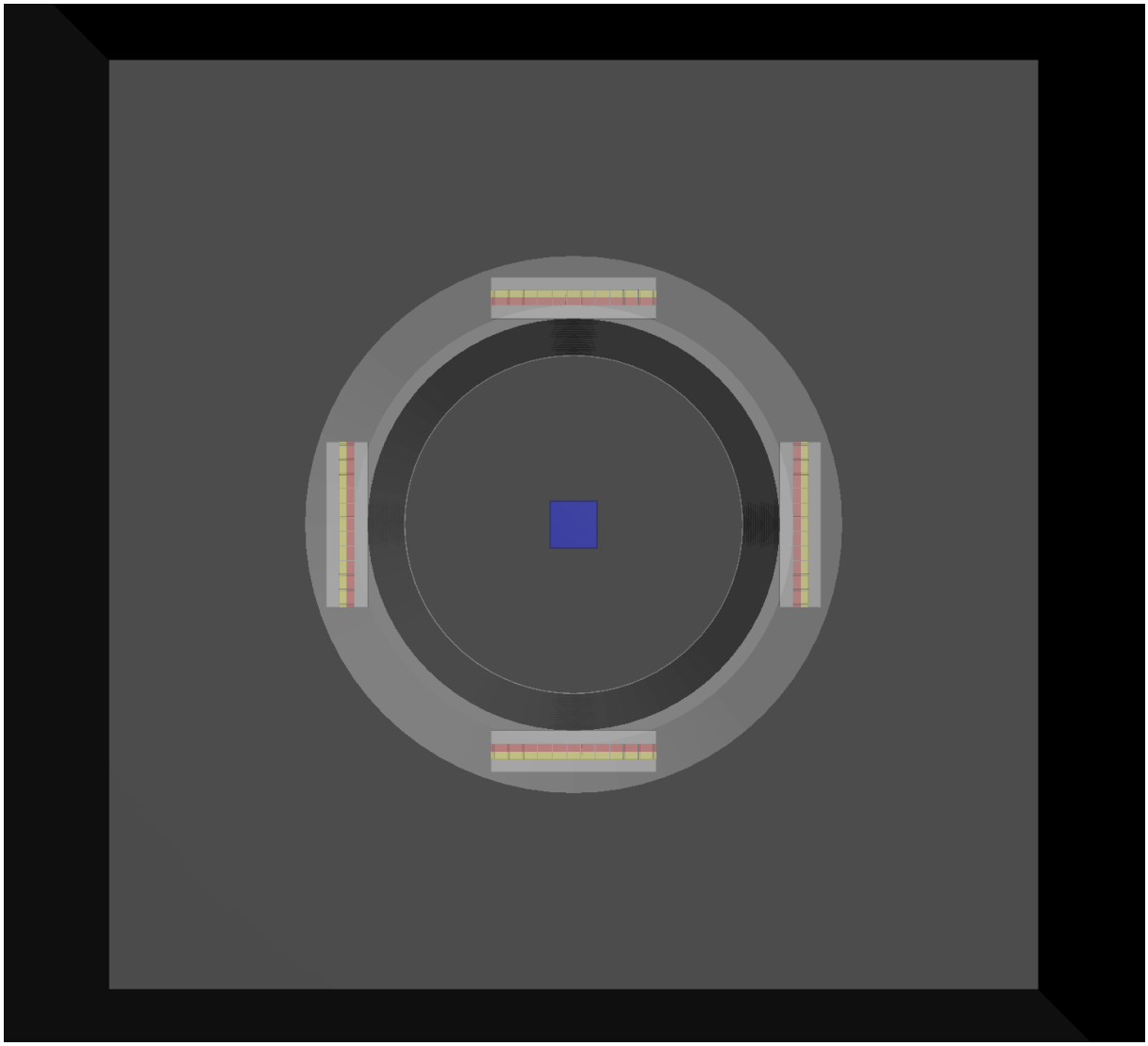
4. RESULTADOS

Este capitulo destina-se a demonstrar os resultados atingidos a partir deste trabalho.

4.1 Visualizador Web de PET Scanners

Ao finalizar o preenchimento dos parâmetros pedidos na página de *scanner* e pressionar o botão submit, outra guia é a aberta no navegador do usuário, contendo a versão simulada do equipamento. A Figura 9 mostra o exemplo de *scanner* renderizado pelo sistema.

Figura 9 - Exemplo de visualização gerada pelo sistema



Além da visualização padrão disponibilizada pelo sistema, o usuário pode interagir com o modelo, i.e., é possível aplicar rotações e *zoom* ao objeto, como mostram as Figuras 10 e 11.

Devido ao alto uso de recursos gráficos e limitações do servidor de hospedagem, a aplicação pode apresentar lentidão nas respostas a interação com o usuário. O carregamento inicial do elemento no navegador, leva em média 50 segundos para ser finalizado.

Um desafio que se fez presente no momento de desenvolvimento, trata-se da baixa escalabilidade do *framework* Django. O mesmo apresenta arquitetura de pastas confusas, pois contém mais de um diretório com o mesmo nome, e seu alto nível de acoplamento entre *front* e *back end* dificultam a criação de funcionalidades de *front end* que utilizem outros recursos além do próprio Django.

Figura 10 - Objeto rotacionado

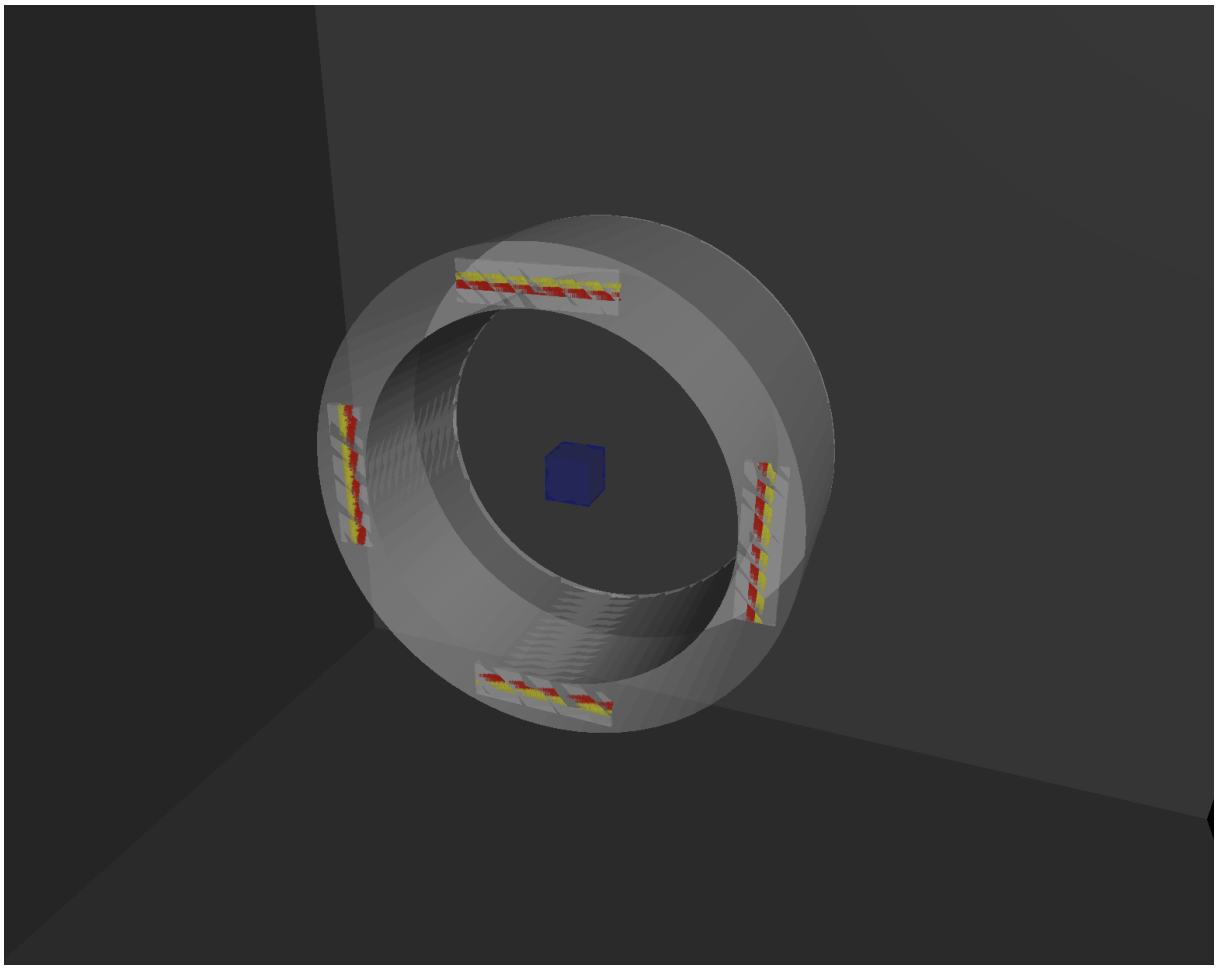
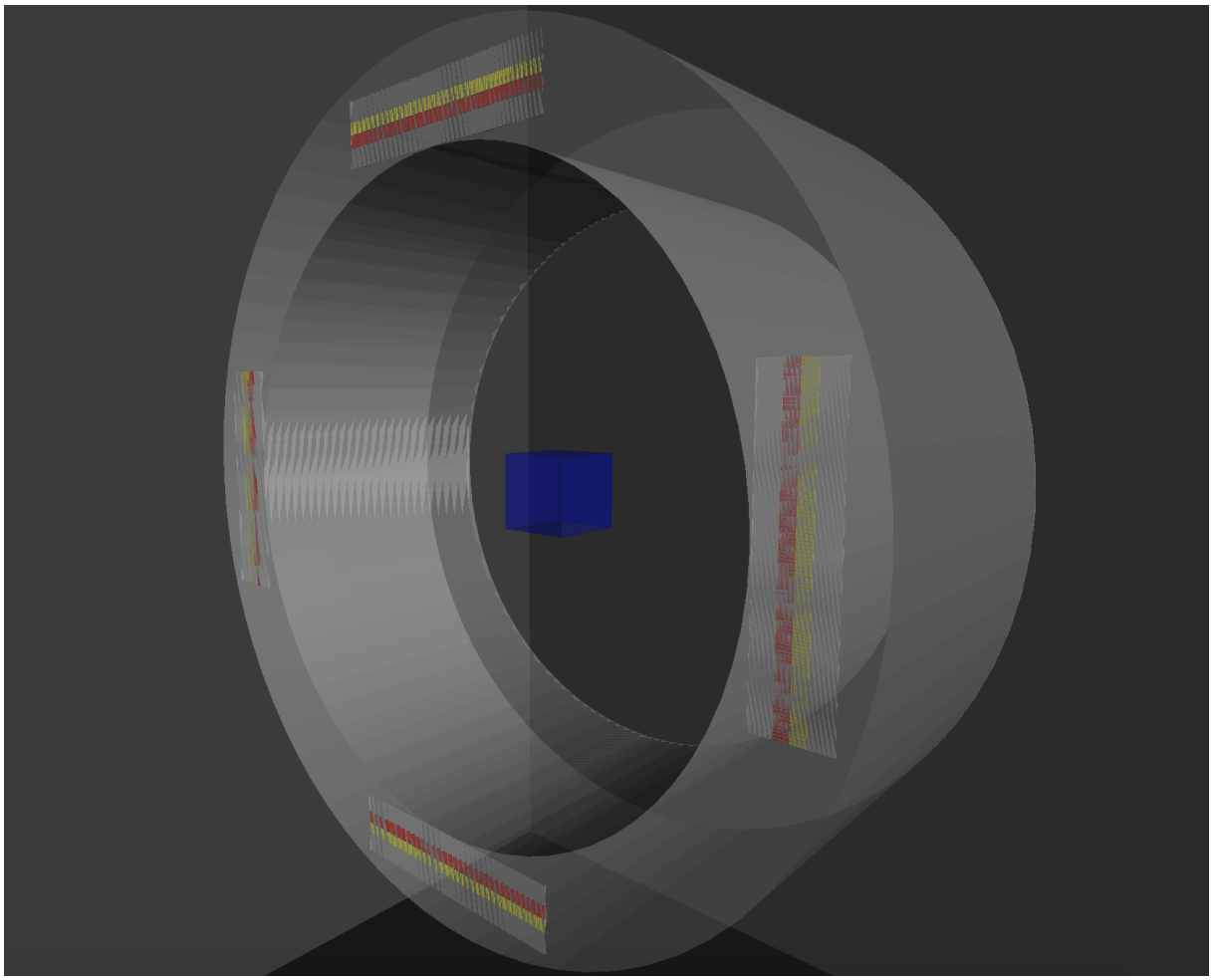


Figura 11 - Visualização do scanner com aproximação



5. CONCLUSÃO

Este trabalho teve como objetivo construir uma ferramenta de visualização de equipamentos PET *scanner* simulados. Tal proposito foi alcançado através da aplicação web construída em Django, JavaScript entre outras tecnologias.

Por meio da nova funcionalidade a pessoa usuária consegue, não só ter uma exibição estática do equipamento simulado, como pode interagir com o mesmo mediante aplicação de rotações e *zoom*, ou seja, uma completa experiência tridimensional.

A disponibilidade de uma aplicação web para essa tarefa melhora significativamente a qualidade no serviço e na pesquisa em medicina nuclear. Além disso, permite o seu uso sem a necessidade de conhecimento prévio de linguagem de programação e códigos de transporte de radiação. Este serviço poderá ser implementado e fornecido por meio de um servidor próprio, tornando-se mais uma utilidade pública disponível para os usuários. Todo o processamento deverá ser feito através deste servidor.

Como tal serviço ainda não existe, a originalidade do projeto agrega a sua importância na área. Por fim, por se tratar de um código de uso global, o estudo e as novas aplicações do código GATE possibilitam oportunidades de contribuição ao desenvolvimento do código e, ainda, o fortalecimento da colaboração internacional.

Todo o processo de concepção e desenvolvimento da aplicação, foi feito levando em consideração a integração com o sistema WebGUIGATE preexistente. Devido a esta restrição, para o desenvolvimento foram usadas tecnologias que não se mostram as opções ótimas dado os requisitos. Como exemplo, destaca-se o *framework* Django, que por vezes tornou o ato do desenvolvimento mais lento, devido ao seu alto nível de acoplamento entre *front* e *back end*.

Pensando nas oportunidades de crescimento deste trabalho, é possível trabalhar no elevado uso de memoria no momento de exibir a simulação na pagina web. Já levando em consideração o sistema como um todo, a migração para outras tecnologias de desenvolvimentos, ou até mesmo uma mudança de paradigmas, são alguns dos caminhos possíveis. Converter a parcela do software que interage com GATE em API, e fazer com que o resto da aplicação apenas consuma a mesma, sanaria algumas questões de acoplamento.

REFERÊNCIAS

BATISTA, Regina Claudia. *Desenvolvimento de interface para ambiente hipermídia voltado ao ensino de geometria sob a ótica da ergonomia e do design gráfico*. . [S.l: s.n.], 2003.

BOMBARDA, Sidney; SOARES JÚNIOR, José; TERRA FILHO, Mário. Estudo do metabolismo da glicose na tuberculose pulmonar ativa utilizando a tomografia por emissão de pósitrons (18F-FDG PET). *Jornal de Pneumologia*, v. 28, n. 5, p. 270–276, 2002. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-35862002000500005&lng=pt&nrm=iso&tlng=pt>.

CORDEIRO, Leanderson Pereira. WebGUIGATE : Aplicação web para construção de macros de simulação de PET scanners usando toolkit GATE. [S.d.].

HORSTMANN, C.S; NECAISE, R.D. *Python for Everyone*. 1. ed. [S.l.]: Wiley Publishing, 2013.

JAN, S. *et al.* GATE: A simulation toolkit for PET and SPECT. *Physics in Medicine and Biology*, 2004.

LJUNGBERG, Michael; STRAND, Sven-Erik; KING, Michael A. *Monte Carlo calculations in nuclear medicine: Applications in diagnostic imaging*. [S.l.]: CRC Press, 2012.

MICHEL M. TER-POGOSSIAN, Marcus E. Raichle and Burton E. Sobel. Positron-Emission Tomography. *Scientific American*, v. 243, n. 4, p. 170–181, 1980.

OPENGATE, Wiki. Users Guide V8.0. 2017. Disponível em: <http://www.opengatecollaboration.org/sites/default/files/GATE-UsersGuideV8.0.pdf>.

SAHA, Gopal B. *Basics of PET imaging: Physics, chemistry, and regulations, third edition*. [S.l: s.n.], 2016.

SELBACH SILVEIRA, Milene; OLIVEIRA PRATES, Raquel. Uma Proposta da

Comunidade para o Ensino de IHC no Brasil. *SBC*, 2007.

TAUBIN, Gabriel *et al.* Geometry coding and VRML. *Proceedings of the IEEE*, 1998.

THREE.JS. *Three.js*. Disponível em: <threejs.org>.

W3C. *HTML & CSS*. Disponível em: <https://www.w3schools.com/>.

W3SCHOOL. *JavaScript*. Disponível em: <https://www.w3schools.com/>.