Relatório Técnico: Implementação do Sistema de Gerenciamento de Atributos em Modelagem Geométrica

Gabriel Paes Gomes
gabriel_pg@id.uff.br

Lucas Alonso Correia de Oliveira
lcorreia@id.uff.br

Universidade Federal Fluminense - UFF
Modelagem Geométrica - TCC00242
Professor: Dr. André Maues Brabo Pereira

14 de Dezembro de 2025

1 Introdução

Este relatório descreve a implementação e a evolução do Modelador Geométrico desenvolvido ao longo da disciplina para suporte de Gerenciamento de Atributos. O objetivo deste módulo é permitir a associação de dados semânticos e físicos (como condições de contorno, materiais, cargas e propriedades definidas pelo usuário) às entidades topológicas (Vértices, Arestas e Faces) de um modelo Half-Edge. O sistema foi desenvolvido sobre a biblioteca HETool, utilizando Python e PySide6 para a interface gráfica, e OpenGL para a visualização.

2 Arquitetura do Sistema

O sistema de atributos segue o padrão MVC (Model-View-Controller) já estabelecido na aplicação, garantindo desacoplamento entre a lógica de dados, o gerenciamento e a visualização.

2.1 Estrutura de Dados (AttribManager)

A classe `AttribManager` atua como o repositório central das definições de atributos. Ela gerencia duas listas principais:

- **Protótipos:** Modelos pré-definidos de atributos complexos (ex: "Support Conditions", "Uniform Load") carregados de um arquivo JSON (*attribprototype.json*). Isso permite a padronização de atributos comuns em engenharia.

- **Atributos Criados:** Uma lista dos atributos instanciados pelo usuário durante a sessão.

A validação dos dados é feita utilizando a biblioteca `jsonschema`, garantindo que a estrutura dos atributos esteja correta antes da criação.

2.2 Controle e Lógica (HeController)

O controlador (`HeController`) é o núcleo da lógica de aplicação. As principais funcionalidades incluem:

- **Criação Dinâmica:** O método `createAndApplyAttribute` gerencia tanto a criação de atributos escalares simples quanto a instanciação de protótipos complexos baseados no nome.
- **Aplicação Direcionada (Targeting):** Implementação de lógica para filtrar a aplicação do atributo baseada no tipo de entidade selecionada (Vertex, Edge ou Face/Patch).
- **Padrão Command:** Todas as operações de atribuição utilizam o padrão Command (`SetAttribute`, `UnSetAttribute`) através da classe `UndoRedo`, permitindo desfazer e refazer ações.

3 Desafios de Implementação e Soluções

Durante o desenvolvimento, identificamos e resolvemos problemas críticos relacionados à persistência e independência dos dados.

3.1 Integração com a HETool: Refatoração vs. Adapter

Um dos desafios significativos foi a integração do modelador com a biblioteca `hetools`. Inicialmente, tentamos uma abordagem utilizando o padrão **Adapter** para isolar as dependências, mas devido à necessidade de uma comunicação mais direta com as estruturas topológicas, decidimos pela **refatoração** completa da integração, garantindo maior fluidez e performance ao sistema.

3.2 Independência de Instâncias (Deep Copy)

O problema identificado foi que, ao aplicar um atributo a múltiplas entidades, o sistema passava uma referência ao objeto, fazendo com que a alteração do valor em uma entidade afetasse todas as outras. **Solução:** Implementamos o uso de `copy.deepcopy()` no método `setAttribute` do controlador. Agora, o sistema cria um clone independente dos dados para cada entidade alvo, garantindo valores isolados.

3.3 Persistência de Dados (HeFile)

Originalmente, o sistema armazenava apenas o nome do atributo, perdendo customizações individuais ao recarregar o arquivo. **Solução:** Reescrevemos os métodos `saveFile` e `loadFile` na classe `HeFile`. Agora, o sistema serializa o dicionário completo de atributos (`entity_attributes`) dentro da estrutura JSON de cada Vértice, Aresta e Face, assegurando a restauração correta de valores específicos.

4 Interface do Usuário

A interface desenvolvida ao longo da disciplina foi projetada de forma bem integrada e intuitiva, facilitando a interação do usuário com o sistema de atributos.

4.1 Diálogo de Criação (AttributeDialog)

O diálogo suporta seleção híbrida (QComboBox editável) para reutilização ou criação, filtragem de entidades por checkboxes (Vertex, Edge, Face) e um seletor de cores integrado.

4.2 Visualizador e Editor (AttributeViewer)

Implementamos um `QDockWidget` lateral interativo que lista os atributos das entidades selecionadas em uma árvore (`QTreeWidget`). O usuário pode realizar edições *in-place* clicando duas vezes no valor, e utilizar o menu de contexto para remover atributos.

5 Visualização Gráfica (OpenGL)

A renderização dos atributos no Canvas fornece feedback visual imediato. A classe `AttribSymbols` gera primitivas geométricas (triângulos para "Support", setas para "Loads") baseadas no tipo de atributo. O método `drawAttributes` no GLCanvas percorre as entidades e desenha os símbolos com escala ajustada automaticamente ao zoom.

6 Próximos Passos

Para as futuras iterações do sistema, planejamos expandir as formas de visualização de atributos, permitindo que o usuário defina como o modelador deve representar os dados na malha. Os principais focos serão:

- **Heatmaps:** Representação de atributos escalares através de gradientes de cor.
- **Texturas Procedurais:** Visualização de propriedades como porosidade através de texturas aplicadas às faces.
- **Customização de Simbologia:** Interface para o usuário definir as regras de representação visual de novos tipos de atributos.

7 Conclusão

O sistema de gerenciamento de atributos implementado oferece uma solução robusta e flexível para a definição de propriedades em modelos geométricos. As modificações realizadas garantiram a integridade dos dados, permitiram a persistência completa e melhoraram a experiência do usuário através de uma interface interativa.