ChatGPT

Tutorial: Criação de Cenas 3D para MuJoCo a partir de Modelos CAD (Fusion 360 e Scans 3D)

Visão Geral e Preparação Inicial

Criar uma cena personalizada no MuJoCo envolve converter modelos 3D (por exemplo, de scans ou CAD) em geometrias adequadas para simulação física. No seu caso, você deseja usar um *scan* 3D do laboratório (obtido via Polycam/LiDAR) como referência para construir uma versão simplificada e *física* do ambiente, a ser utilizada na simulação do robô Unitree G1 no MuJoCo. Em resumo, o **fluxo de trabalho** será:

1. **Obter o modelo 3D de referência** – ex: *scan* do ambiente exportado em formato `.obj` pelo Polycam.
2. **Remodelar no CAD (Fusion 360)** – usar o scan como guia para criar sólidos 3D *fechados* e simplificados (paredes, chão, obstáculos, etc.).
3. **Exportar esses sólidos para formatos compatíveis com o MuJoCo** – tipicamente malhas (`.stl` ou `.obj`) separadas para cada objeto ou grupo de objetos.
4. **Configurar a cena no MuJoCo (arquivo XML MJCF)** – adicionar as malhas como *geoms* (geometrias) estáticas no `worldbody` do modelo MuJoCo (por exemplo, editar o `scene.xml` do Unitree G1), garantindo configurações corretas de colisão, posição e escala.
5. **Testar e ajustar** – carregar a cena no simulador (MuJoCo 3.2.7 no Unitree SDK) e verificar colisões, alinhamento e desempenho, ajustando conforme necessário.

A seguir, detalharemos cada etapa com orientação técnica e melhores práticas, enfatizando pontos importantes como: necessidade de volumes fechados, divisão de objetos concavos em partes convexas, diferenças entre definir a cena via arquivo XML vs. via código Python, e configurações específicas para o MuJoCo/Unitree G1.

1. Considerações sobre Geometrias 3D para MuJoCo

Antes de mergulhar nas ferramentas, é crucial entender **como o MuJoCo trata geometrias importadas** e colisões:

- **Malhas vs. primitivas analíticas**: O MuJoCo permite definir geometrias simples (caixa, esfera, plano, cilindro) via parâmetros de tamanho, ou usar *malhas* personalizadas (arquivos `.stl`, `.obj`, etc.) para formas arbitrárias. No seu caso, as paredes, móveis etc. do laboratório serão melhor representados por malhas derivadas do CAD.

- **Volumetria e fechamento**: Geometrias *planas ou com faces abertas* (sem volume) podem causar problemas na simulação. Por exemplo, uma parede representada apenas por um plano sem espessura pode não colidir corretamente ou pode ser ignorada dependendo da orientação. **Sempre modele com volume** (ex.: uma parede como um paralelepípedo fino, não uma face única) e certifique-se de que não haja buracos nas malhas. Na sua experiência, a primeira cena com ~170

objetos do Polycam falhou possivelmente por conter muitos elementos não sólidos e desconexos, enquanto a segunda cena feita no Blender (com volumes fechados) funcionou bem.

- **Objetos convexos vs. côncavos**: Este é um ponto **muito importante**. O motor de colisão do MuJoCo não lida diretamente com formas côncavas arbitrárias em uma única malha. **Se você importar uma malha que tenha partes côncavas (como um ambiente inteiro com interior oco)**, o MuJoCo internamente usará o **invólucro convexo** dessa malha para detectar colisões [1]. Isso significa que reentrâncias, buracos ou formas de "U" podem ser tratadas como se estivessem preenchidas, causando colisões irreais (por exemplo, um robô *dentro* de um cômodo poderia colidir com o "ar", pois o invólucro convexo do quarto seria um bloco sólido cobrindo a sala toda). A **solução** é **dividir objetos côncavos em múltiplas partes convexas** e importá-las separadamente [1]. Cada parte deve ser uma malha convexa distinta, e todas podem ser agrupadas como partes estáticas do mesmo corpo (no caso de peças fixas do ambiente) [2]. Em outras palavras, em vez de um único `.obj` contendo 170 subobjetos, use **170 arquivos separados** (ou quantos forem necessários) cada um com um volume convexo, ou agrupe superfícies que formam um convexo.

- **Desempenho e número de objetos**: Simular muitos objetos separados aumenta a carga de detecção de colisão, mas objetos estáticos (fixos no mundo) geralmente não causam grande impacto desde que as malhas não sejam excessivamente densas. Prefira **simplificar a geometria** onde possível – por exemplo, não modele detalhes muito pequenos que não influenciam o movimento do robô (cabos, pequenas protuberâncias nas paredes, etc.). Esses detalhes podem ser ignorados ou apenas representados visualmente sem colisão. Lembre-se também de reduzir a resolução das malhas (menos triângulos) se o scan original tiver resolução muito alta; Fusion 360 permite controlar a qualidade na exportação STL.

Com esses conceitos em mente, vamos ao passo a passo.

2. Remodelagem do Ambiente no Fusion 360 usando o Scan 3D

Agora, usando o **Fusion 360** como ferramenta CAD, a ideia é "reconstruir" o ambiente escaneado de forma limpa:

- **Importe o modelo do Polycam**: No Fusion 360, importe o arquivo `.obj` ou outro formato do scan. O Fusion trata malhas como objetos de malha (*mesh*) separados do fluxo de CAD sólido. Você pode **inserir a malha como referência** no espaço de trabalho. Dica: se o arquivo do Polycam contém múltiplos sub-objetos (como 170), o Fusion pode carregá-los como um conjunto de malhas. Pode ser útil agrupá-los ou pelo menos ocultar/exibir por partes para facilitar a visualização.

- **Use o scan como referência para modelar sólidos**: Crie componentes e corpos novos no Fusion 360 e modele **cada elemento arquitetônico ou objeto importante**:

- Para **paredes e piso**: desenhe sketches (esboços) retangulares alinhados às paredes do scan e use *Extrude* para dar espessura (ex.: extrudar uma parede para uma espessura realista, como 0,1m). Garanta que as arestas se encontrem formando um sólido fechado. O chão pode ser um único grande plano extrudado (por exemplo, um retângulo) representando o piso.

- Para **colunas, mesas, obstáculos volumosos**: modele cilindros ou caixas conforme o formato, usando dimensões simplificadas. Não precisa reproduzir cada detalhe pequeno – concentre-se no volume ocupante para colisão.
- **Objetos planos sem volume (ex.: quadros, portas finas)**: dê a eles uma pequena espessura na modelagem. Não deixe nada infinitamente fino.

- **Cheque interferências**: Assegure-se de que seus novos sólidos se encaixam bem ao scan (podem ligeiramente penetrar nas superfícies do scan para cobrir eventuais irregularidades). Depois de modelar tudo, **você pode excluir ou ocultar** o corpo de malha do Polycam – ele terá servido apenas de guia.

- **Topologia e limpeza**: Mantenha a topologia simples – Fusion 360 é um CAD paramétrico, então aproveite para criar formas geométricas perfeitas (planos retos, cilindros uniformes). Isso não só garante colisões mais estáveis, mas também facilita a exportação. Evite superfícies tortuosas ou muito facetadas; em vez disso, aproxime superfícies curvas do scan por cilindros ou prismas regulares, a menos que seja crucial ter detalhes exatos.

- **Sistema de coordenadas e escala**: Tenha atenção ao alinhamento do eixo vertical. O MuJoCo utiliza o eixo **Z como "para cima" (up)** no mundo. Por padrão, no Fusion 360, o eixo *Y* pode ser o "up" dependendo das preferências (é configurável). **Recomenda-se configurar o Fusion para usar Z-up** antes ou durante a modelagem (vá em Preferences -> Default Modeling Orientation -> Z up). Se seu modelo já estiver em Y-up, você pode girar os corpos ou simplesmente lembrar de rotacioná-los na exportação/importação. Além disso, defina a unidade de projeto para **metros (m)** se possível, para coincidir com a escala do MuJoCo e do robô (o Unitree G1 provavelmente está modelado em metros). Caso trabalhe em milímetros (padrão do Fusion para projetos novos), terá que aplicar escala nas malhas exportadas (veremos adiante).

- **Estrutura do modelo CAD**: Coloque os diferentes objetos (paredes, móveis, etc.) como **componentes ou corpos separados** no Fusion. Isso facilita a exportação individual. Você pode também organizar hierarquicamente (por exemplo, um componente "ambiente" contendo subcomponentes "parede1", "mesa", etc.), mas o importante é poder exportar cada parte independente.

3. Exportação das Geometrias do Fusion 360 para o MuJoCo

Com o ambiente modelado em sólidos no Fusion, o próximo passo é exportar esses corpos para arquivos de malha que o MuJoCo possa ler. Siga estas orientações:

- **Exporte um corpo por vez com a mesma origem**: Uma técnica útil para manter o **alinhamento espacial** entre as múltiplas malhas é exportá-las individualmente, mas todas referenciando a mesma origem global. No Fusion 360, você pode fazer o seguinte: **oculte todos os corpos exceto aquele que quer exportar**, então use *File* -> *Export* (ou botão direito no componente -> "Save as STL"/"Save as Mesh"). Isso exportará somente aquele objeto **mas preservando sua posição** em relação à origem do projeto [1]. Repita para cada elemento da cena. Assim, **todas as malhas compartilharão o mesmo sistema de coordenadas de referência**, o que significa que no MuJoCo você poderá posicioná-las usando as coordenadas originais (em muitos casos, `pos="0 0 0"` para

todas, se a origem do CAD foi escolhida apropriadamente para coincidir com, por exemplo, um canto ou o centro do laboratório).

[1] *Dica: O MuJoCo trata cada arquivo de malha como um objeto separado. Se uma malha contém múltiplos sub-objetos, o MuJoCo não irá separá-los; ele considera o conjunto todo como uma geométrica única e usará o invólucro convexo global para colisões. Portanto, exportar cada parte para arquivos distintos (em vez de um .obj único com tudo) garante que você possa definir colisões adequadamente para cada peça.*

- **Escolha do formato**: Recomenda-se exportar em **STL binário** para as malhas estáticas. O STL é amplamente suportado no MuJoCo (formato `.stl` ou `.msh` são mencionados na documentação, e versões recentes também aceitam `.obj`) [3]. O STL não carrega cor/textura, mas isso não é crítico para a simulação física (você pode colorir via XML depois). Se precisar de textura visual, o `.obj` poderia ser usado pois acompanha um `.mtl` – mas geralmente, para cenas simples, cores sólidas definidas no MuJoCo bastam. Ao exportar, **selecione a unidade adequada**: Fusion em metros exporta STL em metros; Fusion em mm exportará em mm – neste caso você terá que escalar dentro do MuJoCo (ex: scale="0.001 0.001 0.001" no asset da malha, para converter mm->m).

- **Nomes e organização**: Salve cada arquivo de malha com um nome identificador (ex: `parede_frontal.stl`, `mesa.stl`, `pilar1.stl`, etc.). Isso facilitará referenciá-los no XML. Centralize todos esses arquivos numa pasta acessível pelo MuJoCo (pode ser na mesma pasta do XML ou subpasta `meshes/`). Lembre que, se estiver usando o *Unitree Mujoco*, ele provavelmente tem um diretório de recursos – você pode colocar suas malhas lá e referenciar o caminho relativo no XML.

- **Verifique a qualidade**: Durante a exportação STL, Fusion permite ajustar a finura da malha (refinamento). Não é necessário uma resolução altíssima para colisão – **malhas mais leves** (menos triângulos) são melhores para performance. Garanta apenas que a forma básica está correta (por exemplo, uma mesa exportada como um bloco retangular – não precisa ter muitos triângulos planos se é toda lisa). Remova detalhes supérfluos antes de exportar para não gerar polígonos desnecessários.

4. Montagem da Cena no MuJoCo via XML (MJCF)

Agora vem a parte de **integrar as malhas exportadas** na definição do mundo do MuJoCo. No contexto do Unitree G1, há um arquivo XML principal (por exemplo, `scene.xml` dentro de `unitree_robots/G1/` no Unitree MuJoCo) que define tanto o robô quanto o cenário. Você pode editar esse arquivo (ou criar um novo) incluindo suas geometrias. Vamos explicar o básico da estrutura MJCF e como adicionar objetos:

- **Estrutura MJCF resumida**: Um arquivo MJCF (MuJoCo XML) tem a forma geral: `<mujoco> ... <asset> ... </asset> ... <worldbody> ... </worldbody> ... </mujoco>`.
- A seção `<asset>` é onde você **declara recursos** (malhas, texturas, materiais) usados na simulação. Aqui vamos registrar cada arquivo STL/OBJ com um nome.

- A seção `<worldbody>` contém a descrição do mundo físico: corpos (*bodies*) e geometrias (*geoms*). O *worldbody* em si é o referencial do mundo (corpo estático raiz). Qualquer geom colocado diretamente dentro de `<worldbody>` **fica fixo no mundo** (sem movimento, a não ser que se aninhe dentro de um `<body>` móvel com articulações).

- **Incluindo as malhas nos assets**: Adicione uma linha para cada arquivo exportado, por exemplo:

```
<asset>
    ...
    <mesh name="parede_frontal_mesh" file="parede_frontal.stl" scale="1 1
1"/>
    <mesh name="mesa_mesh" file="mesa.stl" scale="1 1 1"/>
    <mesh name="pilar1_mesh" file="pilar1.stl" scale="1 1 1"/>
    ...
</asset>
```

Use um atributo `name` único para cada malha (será referenciado no geom). Ajuste `file` com o caminho correto do arquivo (se estiver na mesma pasta do XML, basta o nome; se em subpasta, inclua caminho relativo). Defina `scale` caso seja necessário converter unidades ou redimensionar – por exemplo, se o Fusion exportou em milímetros, use `scale="0.001 0.001 0.001"` para converter para metros [1]. Se já estiver em metros, deixe `scale="1 1 1"` (ou pode omitir, pois 1 é padrão).

- **Criando geoms estáticos no worldbody**: Para cada malha asset, crie uma geom correspondente sob `<worldbody>`. Exemplo:

```
<worldbody>
    ...
    <!-- Parede frontal -->
    <geom name="parede_frontal" type="mesh" mesh="parede_frontal_mesh"
          pos="0 0 0" euler="0 0 0" contype="1" conaffinity="1"
          category="static"/>
    <!-- Mesa -->
    <geom name="mesa" type="mesh" mesh="mesa_mesh"
          pos="0 0 0" euler="0 0 0" contype="1" conaffinity="1"
          rgba="0.7 0.5 0.3 1"/>
    <!-- Pilar -->
    <geom name="pilar1" type="mesh" mesh="pilar1_mesh"
          pos="0 0 0" euler="0 0 0" contype="1" conaffinity="1"/>
    ...
</worldbody>
```

Explicando os atributos utilizados:

- `name`: nome da geometria (opcional, mas útil para depuração).
- `type="mesh"` e `mesh="xxx_mesh"`: indica que a forma vem de uma malha, referenciando o asset pelo nome que você deu.
- `pos="x y z"`: posição da geom no mundo. Se você exportou cada malha já na posição global certa, pode colocar `pos="0 0 0"` para todos, pois as coordenadas dos vértices da malha já incluem a localização relativa. **Importante**: o MuJoCo posiciona a malha tomando o centro (ou

origem) do arquivo como base. Se notar que a peça está deslocada, talvez seja preciso ajustar aqui. Exemplo: se você decidiu que a origem (0,0,0) no Fusion é o canto da sala, mantenha coerência no MuJoCo. Caso contrário, use `pos` para translacionar. Você pode medir distâncias no Fusion ou olhar nas propriedades do corpo para saber a localização de cada peça e pôr aqui.

- `euler="rX rY rZ"`: rotação em Euler XYZ em radianos. Use se precisar rodar a orientação da peça. Ex: se o Fusion estava em Y-up, sua cena pode estar girada 90° ao importar. Você poderia então colocar `euler="-1.5708 0 0"` (–90° em X) para girar tudo de Y-up para Z-up. Se já modelou com Z-up, provavelmente `euler="0 0 0"` está correto.
- `contype` e `conaffinity`: controlam as **máscaras de colisão**. Por padrão, se não mencionar, acho que todo geom herda contype=1 e conaffinity=1 (o que significa que eles colidem entre si e com outros que tenham correspondência). Neste exemplo explicitamos `contype="1" conaffinity="1"` para afirmar que essas geoms interagem colisivamente com o robô (que normalmente também tem contype=1/conaffinity=1 nas geoms do modelo do G1). Se você quisesse, por exemplo, evitar cálculo de colisão entre as **geometrias estáticas entre si** (não é necessário já que todas estão fixas no world, mas o MuJoCo normalmente ignora colisão de geoms no *mesmo* corpo de qualquer forma), poderia usar grupos diferentes. Entretanto, geralmente deixar 1/1 para todos funciona: as geoms do world colidirão com as do robô. MuJoCo não calcula colisão entre geoms que pertençam ao mesmo corpo a menos que explicitamente configurado, e todas essas estão no `worldbody` (que conta como um único corpo estático), então colisões *entre* as paredes e chão não serão computadas (o que é desejado).
- Outros atributos: você pode definir `rgba="R G B A"` para dar cor à peça na visualização (como no exemplo da mesa coloquei marrom claro). Pode também definir `material` se quiser usar um material do asset (com textura, brilho etc.), mas isso é opcional. Atributos de *física* como `friction="μ_slide μ_roll μ_spin"` podem ser ajustados – por exemplo, para o **chão** você pode querer um atrito de ~1.0 (padrão do MuJoCo geralmente é 1). Se não definir, usará default (que costuma ser 1.0 para atrito deslizante).

- **Categoria estática**: no snippet acima usei `category="static"` apenas como informação (essa propriedade não existe literalmente no MJCF – foi ilustrativo). Em vez disso, para o MuJoCo saber que um geom é estático, basta ele estar em um corpo sem juntas móveis. Como estão diretamente sob `worldbody` (que não tem `<freejoint>` nem nada), eles são estáticos. Não coloque `<freejoint/>` nesses objetos, senão eles cairão! (Freejoint indicaria um corpo livre sujeito à gravidade). Portanto, **não** encapsule essas geoms em `<body>` com `<freejoint>`; deixe-as diretamente no world ou num `<body>` fixado a world (sem joints).

- **Integrando com o robô**: Certifique-se de que o robô G1 em si já esteja definido no arquivo. Geralmente, o `scene.xml` do Unitree inclui algo como `<include file="G1.xml"/>` ou define o modelo do robô ali mesmo. De qualquer forma, ao iniciar a simulação, o robô deve aparecer dentro desse ambiente. **Cuide das posições relativas** – por exemplo, se no Fusion você posicionou o robô ou tem a referência de onde ele estaria, talvez queira colocar o modelo do robô naquela posição inicial. No MJCF, o robô provavelmente é um `<body>` com `<freejoint>` posicionado no world. Você pode ajustar a posição inicial do robô (via atributo `pos` do `<body>` base do robô) para colocá-lo, digamos, no chão e próximo de alguma localização desejada. Também certifique que o chão da simulação coincida com z=0 para o robô pousar corretamente. Se suas malhas de piso/paredes foram modeladas com o nível do chão no z=0 (no Fusion), então o robô deve ficar com pés em z=0. Em adição ou como segurança, muitas simulações incluem um geom plano infinito para o chão (tipo `<geom type="plane" size="..."/>`), mas se seu piso for uma malha grande, não

precisa. Só garanta que ele cubra a área necessária e esteja exatamente onde deve estar (você pode ajustar finamente altura se necessário, ex: pos z = 0.001 se precisar evitar z-fighting visual com a malha do chão).

- **Salvar e testar**: Depois de editar o XML com as novas geoms, salve. No *Unitree Mujoco*, veja como executar a simulação – possivelmente rodando o binário `simulate` ou `simulate_python` fornecido, que carregará o `scene.xml` do G1. Se você alterou o `scene.xml` padrão, o simulador já carregará o ambiente novo. Em alternativa, se criou um arquivo separado, verifique se o config aponta para ele (na documentação havia uma variável `ROBOT_SCENE` definindo o caminho do XML da cena).

5. Teste da Simulação e Depuração

Ao rodar a simulação com o novo ambiente, fique atento aos seguintes pontos e dificuldades comuns:

- **Colisões funcionando**: Verifique se o robô colide com o chão e objetos como esperado. Por exemplo, coloque o robô para andar e veja se ele **não atravessa** o piso ou paredes. Se o robô cair infinitamente, pode ser que o chão não esteja detectando colisão – cheque contype/conaffinity e se a malha do chão está correta. Se um objeto específico está "fantasma" (sem colisão), confirme que ele não está marcado com `contype="0"` (o que desligaria colisão) e que sua malha exportou direito (às vezes um STL mal formatado ou com escala errada poderia estar muito pequeno ou grande sem você notar).

- **Objetos caindo ou se movendo**: Todos os objetos do ambiente devem ser estáticos. Se alguma peça do ambiente cair ao iniciar, é sinal de que *acidentalmente* ela foi interpretada como corpo dinâmico. Para resolver, verifique que **não exista** `<joint>` **associado a ela**. Se você, por exemplo, encapsulou uma geom num `<body name="mesa">` que por engano tem um `<freejoint/>`, essa mesa será um corpo solto e vai cair. A solução é remover qualquer joint para mantê-la fixa. Em casos de dúvida, simplesmente coloque as `<geom>` diretamente sob `<worldbody>` sem aninhar em corpos separados (a não ser que você precise montar um grupo de geoms sob um mesmo corpo – o que em estáticos não faz diferença a não ser para organizar XML).

- **Ajustes de posição/orientação**: Se notar desalinhamentos (ex.: uma parede visivelmente fora do lugar comparado ao restante), talvez o pos não devesse ser 0. Nesse caso, ajuste o atributo `pos` daquela geom. É útil habilitar no visualizador do MuJoCo a exibição dos eixos (tem opção de mostrar frames) ou usar a impressão de coordenadas (se `PRINT_SCENE_INFORMATION` estiver ativo no Unitree, ele pode listar poses dos geoms). Em último caso, retorne ao Fusion, coloque o modelo do robô (ou um sistema de coordenadas auxiliar) para ver distâncias. Mas idealmente, com o método de exportação acima, tudo deve encaixar.

- **Questões de escala**: Se o robô parece minúsculo ou gigante em relação ao ambiente, pode ter havido erro de escala. Meça algo simples – por exemplo, a altura de uma porta no lab real vs no sim. Se estiver 1000x maior/menor, é provavelmente a confusão m vs mm. Aplique o `scale` correto no asset das malhas e teste de novo. (Ex.: se você modelou em mm, uma porta de 2m virou "2000" no STL coordenadas; MuJoCo interpretaria como 2000m sem escala – a solução é scale 0.001 no XML).

- **Desempenho**: Veja se a simulação roda suave. Se houver *lag* ou instabilidade, pode ser excesso de polígonos em contato. Objetos concavos mal divididos podem causar contatos múltiplos complexos. O MuJoCo 3.x suporta múltiplos pontos de contato por par de geoms, o que ajuda na estabilidade, mas muitos contatos simultâneos podem pesar. Caso experimente *tremores* ou objetos atravessando, considere simplificar ainda mais alguma malha ou usar representações primitivas. Por exemplo, se o chão é plano, você poderia substituir por um `<geom type="plane">` infinito ou `<geom type="box">` fino grande, em vez de um mesh detalhado do piso. Poderia também excluir colisor de detalhes decorativos (definindo contype 0 só para visual).

- **Visualização**: Ajuste as **câmeras** se necessário. O arquivo XML pode ter uma câmera livre ou predefinida. Como seu cenário é do tamanho de uma sala, posicione a câmera inicial de forma a abranger o ambiente ou siga o robô. Opcionalmente, adicione `<camera>` no XML fixada em algum canto olhando para o centro, para facilitar.

- **Gravidade**: Certifique-se de que a gravidade no MuJoCo está ativada e na direção certa (geralmente `gravity="0 0 -9.81"` no XML). No Unitree sim isso já deve estar configurado. Assim o robô "cai" no chão no início. Se algo fica flutuando, revise isso.

Em suma, itere até a cena estar satisfatória. Um bom sinal é ver o robô interagir fisicamente conforme esperado: pousando com estabilidade no novo chão, colidindo com paredes (e não passando através nem empurrando paredes estáticas, que devem agir como imóveis intransponíveis).

6. Abordagem XML vs. Python para Configuração da Cena

Você perguntou também sobre a diferença entre usar XML e Python para construir a cena. Esclarecendo:

- **XML (MJCF)**: É a forma nativa de descrever modelos no MuJoCo. Você escreve (ou gera) o arquivo de texto com toda a definição do mundo – corpos, geometrias, juntas, sensores, etc. Quando você abre o simulador (por exemplo, usando o binário `simulate` ou via uma chamada `mj_loadXML`), esse arquivo é lido e *compilado* em um modelo interno. **Vantagens**: é declarativo e reproduzível – você pode versionar o XML, editá-lo manualmente ou gerá-lo a partir de outra ferramenta (por exemplo, converter um URDF para MJCF). Já que você "não entendia direito como se faz", a explicação acima de cada tag espero que dê clareza. Com a prática, editar o MJCF se torna mais fácil, especialmente para cenas estáticas (onde é basicamente listar geoms e assets).

- **Python**: Existem duas formas principais de usar Python com MuJoCo:

- **Bindings diretos (mujoco.py)**: A biblioteca oficial (desde MuJoCo 2.0+ open-source) oferece `mujoco` Python API. Com ela, você pode carregar um modelo XML ou até montar um modelo programaticamente (há funções para criar estruturas, mas é avançado). Depois de carregado, você pode usar Python para, por exemplo, *manipular a simulação em tempo real* (aplicar forças, ler sensores, etc.). Mas **adicionar geometrias via Python** depois de compilado o modelo é limitado – o MuJoCo não suporta adicionar/remover corpos em tempo de simulação arbitrariamente. Então normalmente você ainda fornece a descrição da cena via XML, e Python apenas controla o robô ou modifica parâmetros durante a simulação.

- **Gerar o XML via Python**: Outra abordagem é ter um *script* Python que escreve o arquivo XML (por concatenação de strings ou usando alguma biblioteca MJCF generator). Isso automatiza a montagem, útil se você quiser, por exemplo, criar 100 obstáculos randomicamente posicionados – em vez de escrever manualmente, um script gera o MJCF e depois você carrega. Frameworks como o DMControl (DeepMind Control Suite) utilizam módulos MJCF in Python que definem o modelo via código e geram o MJCF internamente.

No contexto do Unitree SDK, eles fornecem duas versões: um simulador C++ e um Python (`simulate_python`). Provavelmente, o Python version usa as bindings para carregar o mesmo `scene.xml` e depois permite controlar o robô via Python. **Não creio que esperam que você construa a cena usando Python**; é mais para controlar o robô (por exemplo, aplicar comandos de movimento através da API do Unitree). Portanto, a *diferença de abordagem* é: usar apenas o XML para definir a estática da cena e o modelo do robô, e Python apenas para lógica de controle vs. tentar definir tudo programaticamente. Para iniciantes, **recomendo fortemente usar o XML para modelar o mundo**, pois é transparente e conforme a documentação. O Python pode complementar, mas não substitui a necessidade de entender a estrutura do modelo.

- **Integração com URDF**: Apenas como nota, há também a possibilidade de usar um URDF (formato do ROS) e deixá-lo ser convertido para MuJoCo MJCF (MuJoCo suporta importar URDF com extensões [4]). Contudo, isso é mais aplicado a robôs articulados. Para o ambiente (paredes, etc.), URDF não adiciona vantagem – você já está construindo direto no MJCF.

Resumindo, **continue montando suas cenas via XML**, pois é o método padrão. Use Python para controlar o robô no ambiente (por exemplo, ler sensores, implementar um loop de controle ou integrar com ROS2, etc., como o Unitree SDK faz).

Caso queira no futuro gerar cenários aleatórios ou modificar parâmetros sem editar XML manualmente, aí sim pense em usar Python para gerar MJCF ou manipular posições (por exemplo, mudar posição de um geom através de `model.geom_pos` antes de iniciar a sim). Mas isso são usos avançados.

7. Dicas Finais e Próximos Passos

Você agora tem um *pipeline* estabelecido: **Scan 3D -> CAD (ajuste) -> Export -> MJCF -> Simulação**. Aplicando isso a um scan LiDAR maior do laboratório real, considere:

- **Planejamento da modelagem**: O lab completo pode ser extenso. Identifique as estruturas fixas principais: paredes, colunas, chão, teto (se for relevante para sensores ou colisão), escadas ou desníveis, etc. Modele esses como geoms estáticos. Depois identifique objetos móveis ou manipuláveis (se forem relevantes para o robô) – estes, se quiser simular fisicamente, devem ser corpos separados possivelmente com dinâmicas (por ex., uma caixa que o robô poderia empurrar seria um `<body>` com massa e geoms e talvez joints se articulado). Se forem apenas obstáculo estático, podem ir junto com cenário estático.
- **Nível de detalhe vs. performance**: Um lab escaneado terá *muito* detalhe (cada cadeira, cada aparelho). Pergunte-se quais são necessários para a simulação. Talvez seja útil simplificar móveis complexos como “caixas” representativas. O robô G1 (humanoide ou quadrúpede? G1 creio que é bípede humanoide?) se beneficiará de ter o terreno e grandes obstáculos bem representados; pequenos objetos podem ser ignorados ou adicionados depois se necessário.

- **Verificação de convexidade**: Após modelar, mentalmente cheque: cada peça é convexa? Se não, você a quebrou em partes? Ex: uma mesa com tampo e pernas forma um conjunto côncavo (porque há espaço vazio entre pernas). Você poderia exportar o tampo e pernas como uma malha única – mas lembre, isso seria concavo (um espaço para passar algo no meio). Se quiser que o robô possa passar por baixo da mesa, *não* combine tampo+pernas numa só malha, pois o MuJoCo verá o conjunto como um bloco convexão englobando o espaço entre as pernas [1]. Em vez disso, modele a mesa com 4 geoms separadas: um para o tampo (plano) e um para cada perna (bastões), todos fixos juntos. Assim, objetos podem passar entre as pernas adequadamente. Todas essas geoms podem pertencer a um mesmo body estático "mesa" para se moverem juntas (mas estático não move de qualquer jeito). Este raciocínio vale para qualquer estrutura com cavidades.

- **Ferramentas auxiliares**: Além do Fusion, às vezes usar o **Blender** ou softwares específicos de edição de malha podem ajudar a decimar ou separar malhas. No seu caso, como você domina Fusion, está redesenhando – o que é ótimo para ter geometria clean. Só tenha em mente que Fusion não é ideal para editar malha poligonais existentes; ele é melhor para redesenhar em CAD. Então para limpeza de um scan denso, talvez use o Fusion apenas para guiar a remodelagem como planejou. Isso deve dar um resultado de qualidade.

- **Documentação e suporte**: Sempre útil consultar a documentação oficial do MuJoCo [1] [3] e fóruns. No seu caso específico (Unitree G1), vale a pena ver se a Unitree forneceu algum exemplo de adicionar objetos ou terrenos além do *terrain_tool* (eles mencionam uma ferramenta para gerar terrenos básicos [5]). Mas como você já conseguiu inserir objetos manualmente, continue expandindo sobre isso.

Por fim, com a cena pronta e o robô simulando no ambiente realista, você pode testar os algoritmos de navegação, visão etc., fazendo um *sim-to-real* mais fiel. Ajuste parâmetros físicos (atrito, restituição) para que o comportamento no simulete aproxime o do mundo real (por exemplo, piso de concreto vs carpete teria atrito diferente).

Boa sorte na construção da cena! Com essa abordagem passo a passo – **scan, CAD cleanup, export, XML integration** – você deverá conseguir um ambiente virtual robusto onde o Unitree G1 possa interagir fisicamente de forma confiável, espelhando o laboratório real. Qualquer anomalia, revise as partes mencionadas (geometria convexa, escalas, etc.), pois geralmente os problemas estarão nessas áreas.

Referências Utilizadas: Conhecimento das restrições de colisão do MuJoCo, conforme discutido no fórum oficial [1], documentação do MuJoCo sobre importação de malhas [3] e experiências práticas de usuários na integração de modelos CAD com MJCF. Essas fontes reforçam a importância de dividir malhas côncavas e manter volumes fechados para obter simulações estáveis e realistas. Boa construção de cena e bons experimentos!

---

[1] [2] Convex decomposition does not add concavity to simulation objects | Legacy MuJoCo Forum
https://www.roboti.us/forum/index.php?threads/convex-decomposition-does-not-add-concavity-to-simulation-objects.13/

[3] XML Reference - MuJoCo Documentation
https://mujoco.readthedocs.io/en/stable/XMLreference.html

4 Is there some kind of CAD program you can design Mujoco MJCF XML files in? : r/robotics
https://www.reddit.com/r/robotics/comments/qt9i95/is_there_some_kind_of_cad_program_you_can_design/

5 GitHub - unitreerobotics/unitree_mujoco
https://github.com/unitreerobotics/unitree_mujoco