

# py360tool: Um Framework para Manipulação de Vídeo 360° com Ladrilhos

Henrique D. Garcia  
henriquedgarcia@gmail.com  
Departamento de Engenharia Elétrica  
Universidade de Brasília  
Brasília, Distrito Federal, Brasil

Marcelo M. Carvalho  
mmcarvalho@txstate.edu  
Ingram School of Engineering  
Texas State University  
San Marcos, Texas, USA

## ABSTRACT

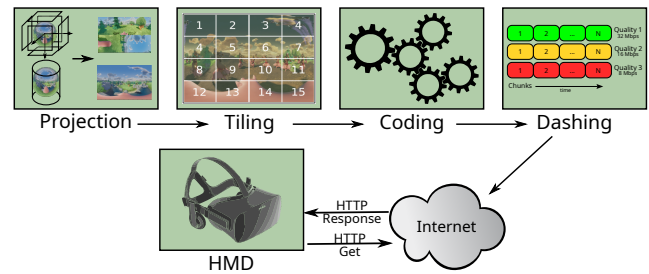
The streaming of 360° videos is one of the most bandwidth-demanding virtual reality (VR) applications, as the video must be encoded in ultra-high resolution to ensure an immersive experience. To optimize its transmission, current approaches partition the spherical video into tiles, which are encoded at different bitrates and selectively delivered, based on the viewing direction of the user (viewport). The complexity of this architecture, which involves viewport prediction, tile selection, bit rate adaptation, and handling of parallel streaming, requires new tools to evaluate quality of experience (QoE) and quality of service (QoS), especially due to its interactive nature and low reproducibility. This work introduces py360tools, a Python library to handle tile-based 360° video streaming. The library automates key client-side tasks, such as spherical projection reconstruction, viewport extraction, and tile selection, facilitating the playback and simulation of streaming sessions. Furthermore, py360tools offers a flexible architecture, enabling efficient analysis of different projections and tiling strategies.

## KEYWORDS

Streaming de Vídeo 360°, Realidade Virtual, Taxa de Bits Adaptativa, Qualidade de Serviço, Qualidade de Experiência.

## 1 INTRODUÇÃO

Dentre as aplicações de Realidade Virtual (RV), o *streaming* de vídeo 360° é o que mais consome largura de banda. Como o campo de visão humano é limitado, é mais eficiente transmitir apenas a região da esfera que o usuário visualiza, chamada de *viewport*. Uma das abordagens promissoras para otimizar essa transmissão é segmentar espacialmente a esfera em ladrilhos (*tiles*) e transmitir apenas os que aparecem no *viewport* usando protocolos como DASH (*Dynamic Adaptive Streaming over HTTP*) ou HLS (*HTTP live streaming*). O cliente então solicita apenas o *chunk* contendo os ladrilhos visíveis naquele instante, adaptando-se à largura de banda disponível. A Figura 1 ilustra o fluxo típico de processamento e transmissão de vídeo 360° via DASH: o vídeo 360° é projetado em um plano, segmentado em ladrilhos e codificado em diferentes qualidades/taxas de bits usando codificadores tradicionais (ex: H.265 ou VP9). Ao chegar no cliente, os *chunks* de diferentes qualidades são remontados em cada quadro da projeção e intercalados de forma que o usuário veja o vídeo sem interrupções

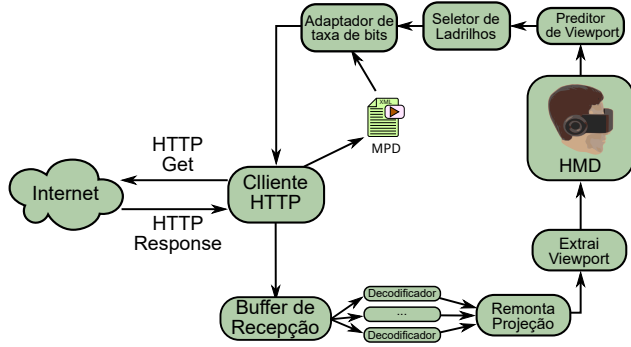


**Figura 1:** Antes de ser transmitido o vídeo é projetado, recortado, comprimido e segmentado em *chunks*. O cliente solicita apenas os *chunks* que aparecem no *viewport*

A organização dos ladrilhos na projeção pode ser feita usando uma extensão do DASH chamada SRD (*Spatial Relationship Description*) [1] ou outros métodos de identificação espacial dos ladrilhos que permite o envio de mais informações como JSON [2]. Porém, em todos os casos, o cliente deve saber onde colocar o ladrilho na esfera. O vídeo é reproduzido em visualizadores de realidade virtual (*Head-Mounted Display* – HMD), que se conectam à CDN pela internet usando conexões sem fio como *Wi-Fi* em uma arquitetura cliente-servidor. O cliente se conecta à CDN via protocolo HTTPS (*HyperText Transfer Protocol Secure*) e, consequentemente, TCP (*Transmission Control Protocol*). O cliente é responsável por controlar a taxa de bits de forma dinâmica, de acordo com a largura de banda disponível.

O funcionamento de um cliente de streaming de vídeo 360° baseado em ladrilhos depende de múltiplos mecanismos, ilustrados na Figura 2. Inicialmente, via HTTP, o cliente requisita o manifesto que contém informações sobre os *chunks*, localização espacial de cada ladrilho na projeção, taxa média de bits dos ladrilhos, respectivos URLs, e outras informações. Os segmentos recebidos são agendados para serem decodificados um pouco antes da sua reprodução, pois o vídeo sem compressão ocupa muita memória. Devido à codificação preditiva, todos os quadros de um *chunk* precisam ser totalmente decodificados para acessar um quadro arbitrário [3]. Antes de serem exibidos ao usuário, os ladrilhos decodificados são montados de volta na projeção e o *viewport* é extraído.

Antes de solicitar o próximo *chunk*, o *viewport* é previsto usando técnicas como regressão linear [5] ou aprendizado de máquina [6] e um mecanismo seletor de ladrilhos prioriza-os de acordo com as configurações do sistema [4]. É possível requisitar apenas os ladrilhos a serem vistos, ou requisitar ladrilhos com diferentes qualidades baseado na probabilidade de serem visualizados pelo usuário. Por fim, o algoritmo de Adaptação da Taxa de Bits (ABR)



**Figura 2: Internamente o cliente pode ser dividido em: preditor de viewport, seletor de ladrilhos, ABR, Cliente HTTP, decodificador, montador de ladrilhos e extrator de viewport. Estes módulos operam de forma sequencial e circular durante a reprodução do vídeo**

define qual taxa de bits deve ser requerida ao servidor com base na prioridade, largura de banda prevista e informações do manifesto, buscando otimizar o uso de recursos e a experiência do usuário.

A natureza interativa do vídeo 360° dificulta a avaliação da qualidade subjetiva, pois a cada reprodução, os requisitos de transmissão dependem da direção para onde o usuário olha, não sendo possível submeter os participantes ao mesmo estímulo. Porém, podemos gravar o movimento de cabeça dos usuários assistindo o vídeo para então simular diferentes situações de transmissão considerando a mesma sequência de viewports. Assim, esta nova sequência de viewport pode ser comparada com a viewport de referência ou ter sua qualidade avaliada de forma subjetiva.

Este trabalho apresenta a biblioteca py360tools, desenvolvida em Python, para manipular vídeos 360° e seus elementos, como projeções, ladrilhos, e viewport. A py360tools encontra-se disponível no GitHub<sup>1</sup>. Esta ferramenta simula diversas etapas de um cliente de vídeo 360° com ladrilhos, servindo como base para um simulador de streaming de vídeo que permite avaliar a qualidade de experiência (*Quality of Service* – QoS) e qualidade de serviço (*Quality of Service* – QoS) sob diferentes condições de hardware e de rede.

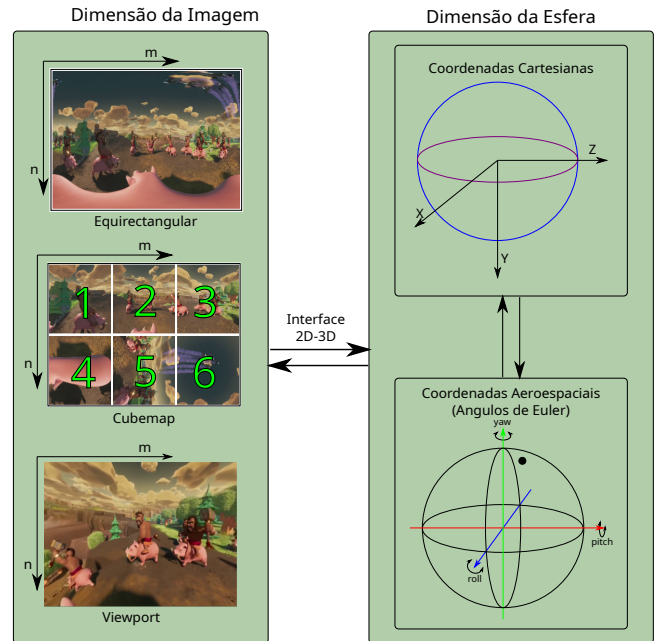
## 2 A BIBLIOTECA PY360TOOLS

A py360tools é um conjunto de ferramentas sob licença GPL-3.0, projetadas para a pesquisa e simulação de streaming de vídeo 360° com ladrilhamento. A biblioteca oferece um conjunto de classes e funções para a manipulação geométrica das imagens, cujas principais funcionalidades são:

- Seleção de Ladrilhos: Identificação e seleção dos ladrilhos que intersectam a viewport do usuário em um dado instante;
- Reconstrução de Projeção: Montagem de uma projeção 2D completa a partir de um conjunto de ladrilhos individuais, que podem possuir diferentes qualidades;

- Extração de Viewport: Geração da janela de visualização do usuário a partir de uma projeção 2D reconstruída;
- Conversão entre Projeções: Mapeamento de pixels entre diferentes formatos de projeção (ex: conversão entre projeções equiretangular e cúbica);
- Conversão de Ladrilhamento: Reorganização de um vídeo em diferentes esquemas de ladrilhamento (ex: alterando o número ou a disposição dos ladrilhos);

A Figura 3 apresenta uma visão geral da arquitetura da biblioteca. O princípio básico da py360tools é uma interface padrão para a conversão de coordenadas entre a dimensão da imagem (projeção 2D) e a dimensão da esfera (3D). Internamente, a biblioteca decodifica os ladrilhos recebidos e os utiliza para reconstruir a projeção. Em seguida, para qualquer operação de mapeamento, as coordenadas dos pixels são primeiro convertidas da projeção de origem para a esfera e, então, da esfera para a projeção de destino. Essa abstração permite o mapeamento flexível de pixels entre diferentes formatos de projeção e facilita operações complexas, como a extração precisa da viewport.



**Figura 3: As transformações ocorrem em duas dimensões. A dimensão da esfera é manipulada em coordenadas cartesianas ou em ângulos de Euler. A dimensão da imagem é onde a codificação é efetuada e as distorções são geradas.**

A dimensão da imagem corresponde a um plano cartesiano 2D onde as coordenadas de cada pixel são representadas por valores inteiros em uma matriz. Nesta representação, todo conteúdo visual, incluindo as projeções esféricas (e.g., equiretangular) e a própria viewport, é inicialmente tratado como uma imagem bidimensional. A dimensão da esfera atua como o espaço canônico para todas as transformações geométricas. Ela é definida por uma esfera de raio unitário centrada na origem de um sistema de coordenadas euclidiano 3D (conforme eixos ilustrados na Figura 3). Ao realizar

<sup>1</sup><https://github.com/henriquedgarcia/py360tools>

uma conversão, os pixels da dimensão da imagem de origem são mapeados para vetores unitários neste espaço 3D. Estes vetores podem então ser remapeados de volta para a dimensão da imagem em um novo formato de projeção. A própria *viewport* é tratada como um caso particular de projeção.

## 2.1 O Modelo de Objeto

Para implementar estes conceitos foram criadas três classes principais (*TileStitcher*, *Projection* e *Viewport*) cujo relacionamento é apresentado no diagrama simplificado da Figura 4.

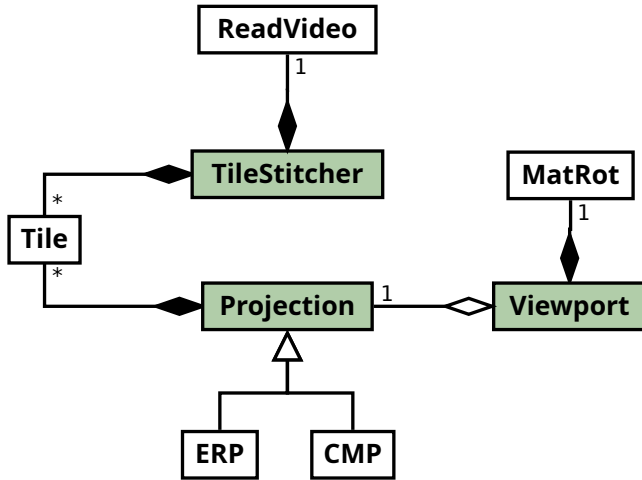


Figura 4: Diagrama de classes simplificado da *py360tools*.

A classe *Projection* é uma classe abstrata para os diferentes formatos de projeção. Ela define operações na dimensão da esfera (3D), enquanto suas classes derivadas são responsáveis por implementar os mapeamentos entre a dimensão da projeção (2D) e da esfera (3D). Atualmente, a *py360tools* oferece implementações para as projeções *Cubemap* e *Equirectangular*. Cada instância de *Projection* possui uma lista de objetos *Tiles*. A classe *Tile* é uma *dataclass* que armazena um índice, posição na projeção, tamanho e o caminho até o arquivo de vídeo associado ao ladrilho.

A classe *TileStitcher* é responsável por reconstruir a projeção 2D a partir de um dicionário de ladrilhos. Ela cria um objeto iterável que sincroniza e costura os ladrilhos fornecidos. Este objeto é inicializado com uma lista de *Tiles*, o ladrilhamento e a resolução da projeção. Os vídeos dos ladrilhos são controlados por diversos objetos *ReadVideo*. Os objetos *ReadVideo* são iteradores que iteram sobre os quadros de um vídeo que implementam funções para acessar o vídeo em uma posição específica e reiniciar o vídeo.

De forma dinâmica, a cada iteração, o objeto *TileStitcher* cria uma imagem vazia da mesma resolução que a projeção. Em seguida, para cada *tile* da lista, extrai um quadro e coloca em suas respectivas posições na projeção. Caso um ladrilho não esteja disponível, a área correspondente na projeção é preenchida com zeros e, por fim, retorna esta imagem.

Objetos da classe *Viewport* fazem Agregação com *Projection* e possuem duas funções principais: extrair o *viewport* de uma projeção e determinar os ladrilhos visíveis. Para extrair o *viewport*, esta classe

implementa uma versão especializada da projeção Gnomônica, parametrizada pelo campo de visão (FoV) do HMD. A orientação da *viewport* é atualizada dinamicamente para refletir a rotação da cabeça do usuário. Este processo é implementado criando a superfície do *viewport* no espaço 3D e aplicando a matriz de rotação nos *pixels* do *viewport* baseada na direção que usuário está olhando. Em seguida convertemos estas novas coordenadas para o domínio da projeção e usamos interpolação para determinar a intensidade do pixel do *viewport*.

A rotação da superfície do *viewport* é definida pelo encadeamento de três rotações em torno dos eixos principais, utilizando os ângulos de Tait-Bryan, uma convenção particular dos ângulos de Euler. Estes ângulos são comumente conhecidos no contexto aeroespacial como (*yaw*), (*pitch*) e (*roll*). A *py360tools* adota como convenção de sinal as rotações no sentido horário sendo ângulos positivos. Esta convenção implica que as rotações dos ângulos de Tait-Bryan deve seguir a sequência Y-X-Z.

Para determinar quais objetos no espaço 3D são visíveis, o campo de visão (*Field of View* - FOV) do HMD é modelado como uma pirâmide de base quadrada cujo o ângulo de abertura determina a base. Esta pirâmide é geometricamente definida pela interseção de quatro planos que se encontram na origem da esfera (Figura 5). Os planos laterais (direito e esquerdo) formam o ângulo horizontal  $FOV_x$ , enquanto os planos superior e inferior definem o ângulo vertical  $FOV_y$ . Cada plano é representado por seu vetor normal, orientado para o interior da pirâmide. Consequentemente, a rotação do pirâmide é calculada aplicando-se a mesma matriz de rotação aos quatro vetores normais.

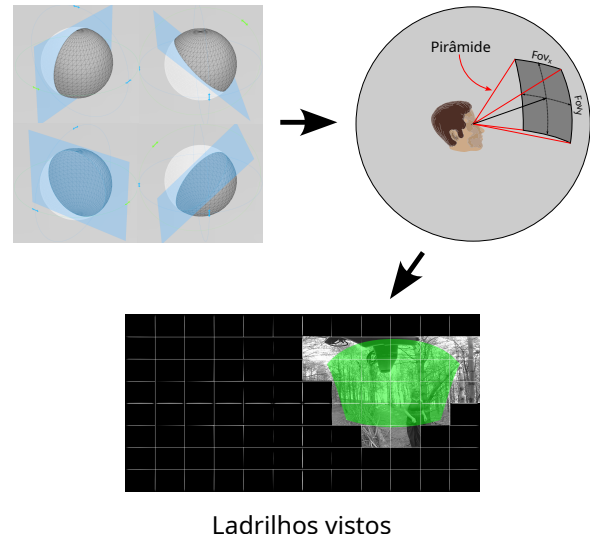


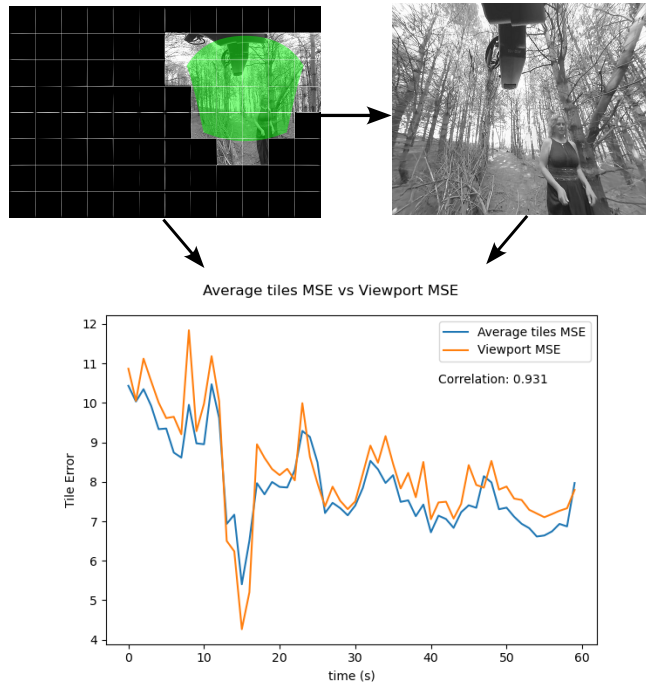
Figura 5: O que é visível (FOV) compreende a região do espaço que faz intersecção com quatro planos concêntricos. Basta um único pixel de um ladrilho estar dentro do FOV para que seja considerado visto.

Um ponto  $p = (x, y, z)$  no espaço 3D pertence à *viewport* se, e somente se, estiver no lado interno de todos os quatro planos. Matematicamente, esta condição é satisfeita se o produto escalar entre o vetor do ponto  $\vec{p}$  e o vetor normal  $N_i = (x_i, y_i, z_i)$  de cada

plano for negativo:  $N_i \cdot \vec{p} \leq 0$ . Para otimizar o processo de seleção de ladrilhos, não é necessário testar cada pixel individualmente. Em vez disso, o teste de pertencimento ao FOV pode ser aplicado de forma eficiente apenas às suas bordas.

Geralmente os dispositivos de realidade virtual, como o Meta Quest 3, possuem um FOV horizontal de aproximadamente  $110^\circ$  e um FOV vertical de  $96^\circ$ <sup>2</sup>. A resolução do *viewport* depende do dispositivo. Quanto maior a resolução do *viewport*, maior deve ser a resolução da projeção. Atualmente, os dispositivos apresentam visores com resolução de 2K (1080p) e suporte limitado a projeções em Ultra Alta Definição (UHD).

Como exemplo de aplicação, a Figura 6 mostra o resultado da extração do *viewport* da projeção preenchida apenas com alguns ladrilhos ao longo de uma sessão. Com base neste *viewport* podemos avaliar a qualidade do que foi transmitido e quais recursos foram utilizados. Ao lado do *viewport* temos um gráfico comparando o MSE do *viewport* de um usuário com o MSE médio dos ladrilhos usados na sua reconstrução. A projeção foi segmentada no formato  $12 \times 8$  e codificada com QP 28 e o *viewport* foi extraído com FOV igual a  $120^\circ \times 90^\circ$ , com resolução de 1080p. Como esperado, o erro introduzido na codificação não produz o mesmo comportamento na projeção.



**Figura 6:** Não é necessário decodificar todos os ladrilhos para extrair o *viewport*. Em vez disso, montamos a projeção apenas com os ladrilhos disponíveis e extraímos seu *viewport*. Assim, podemos avaliar o quanto da distorção produzida pelo codificador ao ladrilho produz distorção no *viewport* exibido ao usuário.

### 3 CONCLUSÕES

Este trabalho apresentou a *py360tools*, uma biblioteca em Python de código aberto para a manipulação e simulação de *streaming* de vídeo  $360^\circ$  baseado em ladrilhos. A ferramenta foi projetada para lidar com a complexidade geométrica de vídeos esféricos, oferecendo funcionalidades essenciais para a pesquisa em Realidade Virtual. Suas principais capacidades incluem a manipulação de ladrilhos, reconstrução de projeções, extração de *viewports* e a conversão entre diferentes formatos de projeção. A arquitetura da *py360tools* abstrai as operações geométricas entre as dimensões 2D (imagem) e 3D (esfera). Isso permite extrair e analisar métricas como taxa de bits, atraso de decodificação e qualidade do vídeo (QoE e QoS) com base nas estratégias de codificação e ladrilhamento. Atualmente, a *py360tools* é capaz de simular um ambiente ideal, com preditor de *viewport* perfeito, ABR básico (qualidade constante), largura de banda infinita e atraso zero. Como trabalho futuro, pretendemos implementar um módulo de ABR e outro para predição de *viewport*. Dessa forma, será possível integrar seu uso a simuladores de rede, como o NS-3, permitindo a simulação do sistema em um ambiente mais realista.

### REFERÊNCIAS

- [1] Savino Dambra, Giuseppe Samela, Lucile Sassatelli, Romaric Pighetti, Ramon Aparicio-Pardo, and Anne Marie Pinna-Déry. 2018. Film Editing: New Levers to Improve VR Streaming. In *Proc. of the 9th ACM Multimedia Systems Conference (MMSys '18)*. Association for Computing Machinery, Inc, 27–39.
- [2] Gabriel de Castro Araújo, Henrique Domingues Garcia, Mylène C.Q. Farias, Ravi Prakash, and Marcelo M. Carvalho. 2025. A 360-degree Video Player for Dynamic Video Editing Applications. *ACM Trans. Multimedia Comput. Commun. Appl.* 21, 9, Article 249 (Sep 2025), 23 pages.
- [3] Xing Liu, Bo Han, Feng Qian, and Matteo Varvello. 2019. LIME: Understanding Commercial  $360^\circ$  Live Video Streaming Services. In *Proceedings of the 10th ACM Multimedia Systems Conference (MMSys '19)*. 154–164.
- [4] Duc V. Nguyen, Huyen T. T. Tran, and Truong Cong Thang. 2020. An Evaluation of Tile Selection Methods for Viewport-Adaptive Streaming of  $360^\circ$ -Degree Video. *ACM Trans. Multimedia Comput. Commun. Appl.* 16, 1, Article 8 (Mar. 2020), 24 pages.
- [5] Feng Qian, Bo Han, Lusheng Ji, and Vijay Gopalakrishnan. 2016. Optimizing  $360^\circ$  Video Delivery over Cellular Networks. In *Proc. of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges, Part of SIGCOMM 2015*. 1–6.
- [6] Junchao Shao, Zishi Li, and Jinpeng Song. 2025. A Hybrid Approach for Accurate Viewport Prediction in  $360^\circ$  Video Streaming. In *IEEE 6th Intl. Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*. IEEE, 1303–1306.

<sup>2</sup><https://www.meta.com/quest/quest-3/>