06. Technical Roadmap & Future Challenges

今後の開発において予定されている、より高度な技術的挑戦について解説します。

1. High Dynamic Range (HDR) Pipeline

現状: 現在のパイプラインは `RGBA8` (各チャンネル8bit整数, 0-255) で動作しています。これは標準的なモニタ表示には十分ですが、光の強度が1.0を超えるような表現（ブルーム、グレア用の輝度情報）はクリッピングされて失われてしまいます。

計画: パイプライン全体を `RGBA16F` (16-bit Half Float) または `RGBA32F` (32-bit Float) に移行します。

- **OpenGL変更点**: テクスチャフォーマットを `GL_RGBA16F` に変更。
- **トーンマッピング**: プレビュー表示時に、HDR値をLDRモニタで表示するためのトーンマップ処理（Reinhard, ACES等）を最終段のシェーダーに追加。
- **OpenEXR対応**: Pythonの `OpenEXR` バインディングを使用し、リニアワークフローに対応した `.exr` 出力をサポート。

2. Undo/Redo System (Command Pattern)

現状: 操作の取り消し機能がありません。

計画: 古典的な **Command Design Pattern** を導入します。

```
class Command(ABC):
    @abstractmethod
    def execute(self): pass
    @abstractmethod
    def undo(self): pass

class ChangeColorCommand(Command):
    def __init__(self, layer, new_color):
        self.layer = layer
        self.old_color = layer.color
        self.new_color = new_color

    def execute(self):
        self.layer.color = self.new_color

    def undo(self):
        self.layer.color = self.old_color
```

これにより、「レイヤー削除」のような破壊的な操作も安全に行えるようになります。

3. Compute Shader Compositing

現状: フラグメントシェーダーとPing-Pong FBOを使用しています。これは互換性が高い反面、レイヤーごとにドローコール（頂点処理〜ラスタライズ）が発生するオーバーヘッドがあります。

計画: OpenGL 4.3+ の **Compute Shader** を導入し、画像合成を純粋な計算タスクとして処理します。

- **利点**: 頂点パイプラインを通さないため高速。共有メモリを活用した最適化が可能。
- **Batch Processing**: 複数の単純なブレンドレイヤー（Normal Blend等）を一度のディスパッチでまとめて処理することで、メモリアクセスを劇的に削減できる可能性があります。

4. Node-Based Compatibility

計画: UIはレイヤーベースを維持しつつ、内部データをBlenderのShader Editorのようなノードグラフとして管理するリファクタリングを検討しています。 これにより、将来的に「作成したMatcapをBlenderのマテリアルノードとしてエクスポートする」機能の実現性が高まります。