Detailed Implementation Plan

Project Structure & Core Architecture

File Structure

Apply

HostProject/

├── Plugins/

│ └── MagicOptimizer/

│ ├── MagicOptimizer.uplugin

│ ├── Resources/

│ │ └── Icon128.png

│ ├── Source/

│ │ └── MagicOptimizer/

│ │ ├── Public/

│ │ │ ├── MagicOptimizer.h

│ │ │ ├── SOptimizerPanel.h

│ │ │ ├── OptimizerSettings.h

│ │ │ ├── OptimizerRun.h

│ │ │ └── PythonBridge.h

│ │ └── Private/

│ │ ├── MagicOptimizerModule.cpp

│ │ ├── SOptimizerPanel.cpp

│ │ ├── OptimizerSettings.cpp

│ │ ├── OptimizerRun.cpp

│ │ └── PythonBridge.cpp

│ └── Content/

│ └── Python/

│ └── magic\_optimizer/

│ ├── \_\_init\_\_.py

│ ├── entry.py

│ ├── io\_csv.py

│ ├── utils.py

│ ├── textures/

│ │ ├── audit.py

│ │ ├── recommend.py

│ │ ├── apply.py

│ │ └── verify.py

│ ├── meshes/

│ │ ├── audit.py

│ │ ├── recommend.py

│ │ ├── apply.py

│ │ └── verify.py

│ └── presets/

│ ├── PC\_High.json

│ ├── Console.json

│ ├── Mobile\_AR.json

│ ├── VR.json

│ ├── VirtualProduction.json

│ ├── Authoring\_Safe.json

│ └── Cinematic.json

Component Diagrams & Class Hierarchies

Component Diagram:

Core Systems: TextureDoctor, MeshDoctor, MaterialDoctor, RuntimeOptimizer, WorldPartitionManager, AudioOptimizer, PackagingManager

Integration Points: IPythonScriptPlugin, UEditorPerProjectUserSettings, IMeshReduction, NaniteTools, AssetRegistry, EditorAssetLibrary, Material Expression Graph Traversal

Class Hierarchies:

MagicOptimizerModule: Main module class.

SOptimizerPanel: UI panel for the optimizer.

OptimizerSettings: Configuration settings for optimization.

PythonBridge: Bridge to run Python scripts.

OptimizerRun: Orchestration of the optimization phases.

Data Flow Diagrams:

User interacts with SOptimizerPanel.

Settings are read from OptimizerSettings.

OptimizerRun orchestrates audit, recommend, apply, and verify phases.

PythonBridge runs Python scripts for each phase.

Results are reported back to the UI.

Core System Implementation

OptimizerSettings

OptimizerSettings.h

Apply

#pragma once

#include "CoreMinimal.h"

#include "UObject/NoExportTypes.h"

#include "OptimizerSettings.generated.h"

UENUM()

enum class EOptimizerCategory : uint8 {

Textures UMETA(DisplayName = "Textures"),

Meshes UMETA(DisplayName = "Meshes"),

Materials UMETA(DisplayName = "Materials"),

Levels UMETA(DisplayName = "Levels")

};

UCLASS(config=EditorPerProjectUserSettings)

class MAGICOPTIMIZER\_API UOptimizerSettings : public UObject

{

GENERATED\_BODY()

public:

UOptimizerSettings();

static UOptimizerSettings\* Get();

// Scope settings

UPROPERTY(Config, EditAnywhere, Category = "Scope")

bool bUseSelection;

UPROPERTY(Config, EditAnywhere, Category = "Scope")

FString IncludePathsCsv;

UPROPERTY(Config, EditAnywhere, Category = "Scope")

FString ExcludePathsCsv;

UPROPERTY(Config, EditAnywhere, Category = "Global Run")

bool bDryRun;

UPROPERTY(Config, EditAnywhere, Category = "Global Run")

bool bSaveAssets;

UPROPERTY(Config, EditAnywhere, Category = "Global Run")

bool bAutoCheckout;

UPROPERTY(Config, EditAnywhere, Category = "Global Run")

bool bCloseOpenEditors;

UPROPERTY(Config, EditAnywhere, Category = "Global Run")

bool bVerifyAfterWrite;

UPROPERTY(Config, EditAnywhere, Category = "Global Run")

int32 MaxChangesPerRun;

// Target profile

UPROPERTY(Config, EditAnywhere, Category = "Target Profile")

FString TargetProfile;

// Category mask

UPROPERTY(Config, EditAnywhere, Category = "Target Profile")

uint8 CategoryMask;

// Conservative gates

UPROPERTY(Config, EditAnywhere, Category = "Conservative Gates")

bool bConservativeMode;

UPROPERTY(Config, EditAnywhere, Category = "Conservative Gates")

TArray<FString> ApplyOnlyKeys;

// Reporting

UPROPERTY(Config, EditAnywhere, Category = "Reporting")

FString ReportDirName;

// Schema version for migrations

UPROPERTY(Config)

int32 SchemaVersion;

};

OptimizerSettings.cpp

Apply

#include "OptimizerSettings.h"

#include "UObject/SaveGame.h"

#include "Engine/Engine.h"

UOptimizerSettings::UOptimizerSettings()

{

bUseSelection = false;

bDryRun = true;

bSaveAssets = true;

bAutoCheckout = false;

bCloseOpenEditors = true;

bVerifyAfterWrite = true;

MaxChangesPerRun = 1000;

TargetProfile = TEXT("PC\_High");

CategoryMask = EOptimizerCategory::Textures | EOptimizerCategory::Meshes | EOptimizerCategory::Materials | EOptimizerCategory::Levels;

bConservativeMode = true;

ReportDirName = TEXT("OptimizorAudit");

SchemaVersion = 1;

}

UOptimizerSettings\* UOptimizerSettings::Get()

{

static UOptimizerSettings\* Settings = nullptr;

if (!Settings)

{

Settings = Cast<UOptimizerSettings>(UGameInstance::GetSubsystem<UObject>(GetTransientPackage(), UOptimizerSettings::StaticClass()));

if (!Settings)

{

Settings = NewObject<UOptimizerSettings>(GetTransientPackage(), UOptimizerSettings::StaticClass());

}

}

return Settings;

}

PythonBridge

PythonBridge.h

Create file

#pragma once

#include "CoreMinimal.h"

#include "UObject/NoExportTypes.h"

#include "PythonBridge.generated.h"

UCLASS()

class MAGICOPTIMIZER\_API UPythonBridge : public UObject

{

GENERATED\_BODY()

public:

static bool RunPythonTool(const FString& JsonPath, const FString& Phase, const FString Category, FString& OutStdErr);

};

PythonBridge.cpp

Create file

#include "PythonBridge.h"

#include "PythonScriptPlugin/Public/PythonScriptPlugin.h"

bool UPythonBridge::RunPythonTool(const FString& JsonPath, const FString& Phase, const FString Category, FString& OutStdErr)

{

FString Command = FString::Printf(TEXT("import importlib, magic\_optimizer.entry as e; importlib.reload(e); e.run(r\"%s\", phase=\"%s\", category=\"%s\")"), \*JsonPath, \*Phase, \*Category);

bool bSuccess = FIPythonScriptPlugin::Get()->ExecPythonCommand(\*Command, OutStdErr);

return bSuccess;

}

OptimizerRun

OptimizerRun.h

Create file

#pragma once

#include "CoreMinimal.h"

#include "UObject/NoExportTypes.h"

#include "OptimizerRun.generated.h"

UCLASS()

class MAGICOPTIMIZER\_API UOptimizerRun : public UObject

{

GENERATED\_BODY()

public:

void Run(const FString& Phase, const TArray<FString>& Categories);

private:

FString NewRunId();

void WriteConfigJson(const FString& ConfigPath);

void ExecutePhase(const FString& JsonPath, const FString& Phase, const FString Category);

};

OptimizerRun.cpp

Create file

#include "OptimizerRun.h"

#include "OptimizerSettings.h"

#include "PythonBridge.h"

void UOptimizerRun::Run(const FString& Phase, const TArray<FString>& Categories)

{

UOptimizerSettings\* Settings = UOptimizerSettings::Get();

FString ConfigPath = FPaths::ProjectSavedDir() + TEXT("Optimizor/config.json");

// Create new run ID and output directory

FString RunId = NewRunId();

FString OutputDir = FPaths::ProjectSavedDir() + TEXT("Optimizor/History/") + RunId;

IFileManager::Get().MakeDirectory(\*OutputDir, true);

// Flatten settings to JSON

TSharedPtr<FJsonObject> JsonObject = MakeShareable(new FJsonObject);

JsonObject->SetStringField(TEXT("run\_id"), RunId);

JsonObject->SetStringField(TEXT("phase"), Phase);

JsonObject->SetStringField(TEXT("target\_profile"), Settings->TargetProfile);

// Categories

TArray<TSharedPtr<FJsonValue>> CategoryArray;

for (const FString& Category : Categories)

{

CategoryArray.Add(MakeShareable(new FJsonValueString(Category)));

}

JsonObject->SetArrayField(TEXT("categories"), CategoryArray);

// Scope

TSharedPtr<FJsonObject> ScopeObject = MakeShareable(new FJsonObject);

ScopeObject->SetBoolField(TEXT("use\_selection"), Settings->bUseSelection);

ScopeObject->SetStringField(TEXT("include"), Settings->IncludePathsCsv);

ScopeObject->SetStringField(TEXT("exclude"), Settings->ExcludePathsCsv);

JsonObject->SetObjectField(TEXT("scope"), ScopeObject);

// Safety

TSharedPtr<FJsonObject> SafetyObject = MakeShareable(new FJsonObject);

SafetyObject->SetBoolField(TEXT("dry\_run"), Settings->bDryRun);

SafetyObject->SetIntegerField(TEXT("max\_changes"), Settings->MaxChangesPerRun);

SafetyObject->SetBoolField(TEXT("auto\_checkout"), Settings->bAutoCheckout);

SafetyObject->SetBoolField(TEXT("close\_open\_editors"), Settings->bCloseOpenEditors);

SafetyObject->SetBoolField(TEXT("verify\_after\_write"), Settings->bVerifyAfterWrite);

SafetyObject->SetBoolField(TEXT("save\_assets"), Settings->bSaveAssets);

JsonObject->SetObjectField(TEXT("safety"), SafetyObject);

// Gates

TArray<TSharedPtr<FJsonValue>> ApplyOnlyArray;

for (const FString& Key : Settings->ApplyOnlyKeys)

{

ApplyOnlyArray.Add(MakeShareable(new FJsonValueString(Key)));

}

TSharedPtr<FJsonObject> GatesObject = MakeShareable(new FJsonObject);

GatesObject->SetBoolField(TEXT("conservative\_mode"), Settings->bConservativeMode);

GatesObject->SetArrayField(TEXT("apply\_only"), ApplyOnlyArray);

JsonObject->SetObjectField(TEXT("gates"), GatesObject);

// Paths

TSharedPtr<FJsonObject> PathsObject = MakeShareable(new FJsonObject);

PathsObject->SetStringField(TEXT("project\_dir"), FPaths::ProjectDir());

PathsObject->SetStringField(TEXT("output\_dir"), OutputDir);

JsonObject->SetObjectField(TEXT("paths"), PathsObject);

// Write JSON to config path

FString JsonStr;

TSharedRef<TJsonWriter<>> Writer = TJsonWriterFactory<>::Create(&JsonStr);

FJsonSerializer::Serialize(JsonObject.ToSharedRef(), Writer);

FString Error;

if (!FFileHelper::SaveStringToFile(\*JsonStr, \*ConfigPath))

{

UE\_LOG(LogTemp, Warning, TEXT("Failed to write config JSON"));

return;

}

// Execute each phase for selected categories

for (const FString& Category : Categories)

{

ExecutePhase(ConfigPath, Phase, Category);

}

}

FString UOptimizerRun::NewRunId()

{

FDateTime Now = FDateTime::UtcNow();

return Now.ToString(TEXT("%Y%m%d\_%H%M%S"));

}

void UOptimizerRun::WriteConfigJson(const FString& ConfigPath)

{

// This is handled in the Run function already.

}

void UOptimizerRun::ExecutePhase(const FString& JsonPath, const FString& Phase, const FString Category)

{

FString StdErr;

bool bSuccess = UPythonBridge::RunPythonTool(JsonPath, Phase, Category, StdErr);

if (!bSuccess)

{

UE\_LOG(LogTemp, Warning, TEXT("Failed to execute phase %s for category %s: %s"), \*Phase, \*Category, \*StdErr);

}

}

Python Integration

Entry Point

entry.py

Apply

import os

import json

from pathlib import Path

# Import category modules

from . import io\_csv

from .utils import resolve\_enum

from .textures import audit as textures\_audit, recommend as textures\_recommend, apply as textures\_apply, verify as textures\_verify

from .meshes import audit as meshes\_audit, recommend as meshes\_recommend, apply as meshes\_apply, verify as meshes\_verify

def run(config\_path, phase, category):

with open(config\_path, 'r') as f:

config = json.load(f)

output\_dir = config['paths']['output\_dir']

os.makedirs(output\_dir, exist\_ok=True)

if category == "Textures":

if phase == "audit":

return textures\_audit.run(config, output\_dir)

elif phase == "recommend":

return textures\_recommend.run(config, output\_dir)

elif phase == "apply":

return textures\_apply.run(config, output\_dir)

elif phase == "verify":

return textures\_verify.run(config, output\_dir)

elif category == "Meshes":

if phase == "audit":

return meshes\_audit.run(config, output\_dir)

elif phase == "recommend":

return meshes\_recommend.run(config, output\_dir)

elif phase == "apply":

return meshes\_apply.run(config, output\_dir)

elif phase == "verify":

return meshes\_verify.run(config, output\_dir)

return {

"phase": phase,

"category": category,

"scanned": 0,

"changed": 0,

"skipped": 0,

"errors": 0,

"artifacts": {},

"summary": {}

}

CSV Utilities

io\_csv.py

Apply

import csv

import os

from pathlib import Path

def read\_csv(file\_path, encoding='utf-8'):

if not os.path.exists(file\_path):

return []

data = []

try:

with open(file\_path, 'r', newline='', encoding=encoding) as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

data.append(row)

except Exception as e:

print(f"Error reading CSV {file\_path}: {e}")

return []

return data

def write\_csv(file\_path, fieldnames, rows, encoding='utf-8'):

os.makedirs(os.path.dirname(file\_path), exist\_ok=True)

try:

with open(file\_path, 'w', newline='', encoding=encoding) as csvfile:

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writeheader()

writer.writerows(rows)

except Exception as e:

print(f"Error writing CSV {file\_path}: {e}")

def validate\_csv\_header(file\_path, expected\_fields):

if not os.path.exists(file\_path):

return False

try:

with open(file\_path, 'r', newline='', encoding='utf-8') as csvfile:

reader = csv.reader(csvfile)

header = next(reader, [])

return set(header) >= set(expected\_fields)

except Exception:

return False

def get\_csv\_field\_value(row, field\_name, default=None):

return row.get(field\_name, default)

Utils

utils.py

Apply

import os

from pathlib import Path

def resolve\_enum(enum\_cls, name, default=None):

key = str(name).lower().replace("\_","")

for attr in dir(enum\_cls):

if not attr.startswith("\_\_"):

if attr.lower().replace("\_","")==key:

return getattr(enum\_cls, attr)

return default

def get\_project\_root():

return os.getcwd()

def ensure\_directory\_exists(path):

os.makedirs(path, exist\_ok=True)

def format\_path\_for\_unreal(path):

formatted = path.replace('\\', '/')

return formatted

def get\_file\_size(file\_path):

try:

return os.path.getsize(file\_path)

except:

return 0

def is\_valid\_asset\_path(path):

return path and path.startswith('/Game/') and not path.endswith('/')

UI Components

SOptimizerPanel

SOptimizerPanel.h

Create file

#pragma once

#include "CoreMinimal.h"

#include "Widgets/SCompoundWidget.h"

class SOptimizerPanel : public SCompoundWidget

{

SLATE\_BEGIN\_ARGS(SOptimizerPanel) {}

SLATE\_END\_ARGS()

public:

void Construct(const FArguments& InArgs);

private:

FReply OnAuditClicked();

FReply OnApplyClicked();

ECheckBoxState IsTexturesChecked() const;

void OnTexturesChanged(ECheckBoxState NewState);

ECheckBoxState IsMeshesChecked() const;

void OnMeshesChanged(ECheckBoxState NewState);

ECheckBoxState IsMaterialsChecked() const;

void OnMaterialsChanged(ECheckBoxState NewState);

ECheckBoxState IsLevelsChecked() const;

void OnLevelsChanged(ECheckBoxState NewState);

ECheckBoxState IsUseSelectionChecked() const;

void OnUseSelectionChanged(ECheckBoxState NewState);

FString GetIncludePaths() const;

void OnIncludePathsChanged(const FText& NewText);

FString GetExcludePaths() const;

void OnExcludePathsChanged(const FText& NewText);

TSharedRef<SWidget> OnGenerateProfileComboItem(FString Item) const;

void OnProfileSelected(TSharedPtr<FString> SelectedItem, ESelectInfo::Type SelectInfo);

FText GetCurrentProfileText() const;

TSharedRef<SWidget> OnGenerateRunModeComboItem(FString Item) const;

void OnRunModeSelected(TSharedPtr<FString> SelectedItem, ESelectInfo::Type SelectInfo);

FText GetCurrentRunModeText() const;

private:

UOptimizerSettings\* OptimizerSettings;

TArray<FString> TargetProfiles;

TArray<FString> RunModes;

};

SOptimizerPanel.cpp

Apply

#include "SOptimizerPanel.h"

#include "Widgets/Layout/SBox.h"

#include "Widgets/Text/STextBlock.h"

#include "Widgets/Input/SButton.h"

#include "Widgets/Input/SCheckBox.h"

#include "Widgets/Input/SEditableTextBox.h"

#include "Widgets/Layout/SGridPanel.h"

#include "Widgets/Layout/SVerticalBox.h"

#include "Widgets/Layout/SUniformGridPanel.h"

#include "Framework/Notifications/SNotificationList.h"

#include "EditorStyleSet.h"

#include "OptimizerSettings.h"

void SOptimizerPanel::Construct(const F

keep going

Agent

Qwen 2.5 Coder 32b

Alt⏎ Active file

⏎ Enter

Certainly! Let's continue implementing the SOptimizerPanel class and ensure it integrates well with the rest of the plugin.

SOptimizerPanel Implementation

Header File (SOptimizerPanel.h)

We'll keep the header file mostly as is, ensuring all necessary functions are declared.

SOptimizerPanel.h

Create file

#pragma once

#include "CoreMinimal.h"

#include "Widgets/SCompoundWidget.h"

class SOptimizerPanel : public SCompoundWidget

{

SLATE\_BEGIN\_ARGS(SOptimizerPanel) {}

SLATE\_END\_ARGS()

public:

void Construct(const FArguments& InArgs);

private:

FReply OnAuditClicked();

FReply OnApplyClicked();

ECheckBoxState IsTexturesChecked() const;

void OnTexturesChanged(ECheckBoxState NewState);

ECheckBoxState IsMeshesChecked() const;

void OnMeshesChanged(ECheckBoxState NewState);

ECheckBoxState IsMaterialsChecked() const;

void OnMaterialsChanged(ECheckBoxState NewState);

ECheckBoxState IsLevelsChecked() const;

void OnLevelsChanged(ECheckBoxState NewState);

ECheckBoxState IsUseSelectionChecked() const;

void OnUseSelectionChanged(ECheckBoxState NewState);

FString GetIncludePaths() const;

void OnIncludePathsChanged(const FText& NewText);

FString GetExcludePaths() const;

void OnExcludePathsChanged(const FText& NewText);

TSharedRef<SWidget> OnGenerateProfileComboItem(FString Item) const;

void OnProfileSelected(TSharedPtr<FString> SelectedItem, ESelectInfo::Type SelectInfo);

FText GetCurrentProfileText() const;

TSharedRef<SWidget> OnGenerateRunModeComboItem(FString Item) const;

void OnRunModeSelected(TSharedPtr<FString> SelectedItem, ESelectInfo::Type SelectInfo);

FText GetCurrentRunModeText() const;

private:

UOptimizerSettings\* OptimizerSettings;

TArray<FString> TargetProfiles;

TArray<FString> RunModes;

};

Source File (SOptimizerPanel.cpp)

Let's complete the implementation of SOptimizerPanel.

SOptimizerPanel.cpp

Apply

#include "SOptimizerPanel.h"

#include "Widgets/Layout/SBox.h"

#include "Widgets/Text/STextBlock.h"

#include "Widgets/Input/SButton.h"

#include "Widgets/Input/SCheckBox.h"

#include "Widgets/Input/SEditableTextBox.h"

#include "Widgets/Layout/SGridPanel.h"

#include "Widgets/Layout/SVerticalBox.h"

#include "Widgets/Layout/SUniformGridPanel.h"

#include "Framework/Notifications/SNotificationList.h"

#include "EditorStyleSet.h"

#include "OptimizerSettings.h"

void SOptimizerPanel::Construct(const FArguments& InArgs)

{

OptimizerSettings = UOptimizerSettings::Get();

// Initialize profiles

TargetProfiles.Add(TEXT("PC\_High"));

TargetProfiles.Add(TEXT("Console"));

TargetProfiles.Add(TEXT("Mobile\_AR"));

TargetProfiles.Add(TEXT("VR"));

TargetProfiles.Add(TEXT("VirtualProduction"));

TargetProfiles.Add(TEXT("Cinematic"));

TargetProfiles.Add(TEXT("Authoring\_Safe"));

RunModes.Add(TEXT("Audit"));

RunModes.Add(TEXT("Apply"));

ChildSlot

[

SNew(SVerticalBox)

// Header row

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(4.0f)

[

SNew(SGridPanel)

.FillColumn(1, 1.0f)

+ SGridPanel::Slot(0, 0)

.Padding(2.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Target Profile")))

]

+ SGridPanel::Slot(1, 0)

.Padding(2.0f)

[

SNew(SComboBox<FString>)

.OptionsSource(&TargetProfiles)

.OnGenerateWidget(this, &SOptimizerPanel::OnGenerateProfileComboItem)

.OnSelectionChanged(this, &SOptimizerPanel::OnProfileSelected)

.Content()

[

SNew(STextBlock)

.Text(this, &SOptimizerPanel::GetCurrentProfileText)

]

]

+ SGridPanel::Slot(0, 1)

.Padding(2.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Categories")))

]

+ SGridPanel::Slot(1, 1)

.Padding(2.0f)

[

SNew(SUniformGridPanel)

.SlotPadding(4.0f)

+ SUniformGridPanel::Slot(0, 0)

[

SNew(SCheckBox)

..IsChecked(this, &SOptimizerPanel::IsTexturesChecked)

.OnCheckStateChanged(this, &SOptimizerPanel::OnTexturesChanged)

.Content()

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Textures")))

]

]

+ SUniformGridPanel::Slot(1, 0)

[

SNew(SCheckBox)

..IsChecked(this, &SOptimizerPanel::IsMeshesChecked)

.OnCheckStateChanged(this, &SOptimizerPanel::OnMeshesChanged)

.Content()

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Meshes")))

]

]

+ SUniformGridPanel::Slot(0, 1)

[

SNew(SCheckBox)

..IsChecked(this, &SOptimizerPanel::IsMaterialsChecked)

.OnCheckStateChanged(this, &SOptimizerPanel::OnMaterialsChanged)

.Content()

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Materials")))

]

]

+ SUniformGridPanel::Slot(1, 1)

[

SNew(SCheckBox)

..IsChecked(this, &SOptimizerPanel::IsLevelsChecked)

.OnCheckStateChanged(this, &SOptimizerPanel::OnLevelsChanged)

.Content()

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Levels")))

]

]

]

+ SGridPanel::Slot(0, 2)

.Padding(2.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Scope")))

]

+ SGridPanel::Slot(1, 2)

.Padding(2.0f)

[

SNew(SVerticalBox)

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(2.0f)

[

SNew(SCheckBox)

..IsChecked(this, &SOptimizerPanel::IsUseSelectionChecked)

.OnCheckStateChanged(this, &SOptimizerPanel::OnUseSelectionChanged)

.Content()

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Use Selection")))

]

]

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(2.0f)

[

SNew(SEditableTextBox)

.Text(this, &SOptimizerPanel::GetIncludePaths)

.OnTextChanged(this, &SOptimizerPanel::OnIncludePathsChanged)

.HintText(FText::FromString(TEXT("Include paths (comma separated)")))

]

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(2.0f)

[

SNew(SEditableTextBox)

.Text(this, &SOptimizerPanel::GetExcludePaths)

.OnTextChanged(this, &SOptimizerPanel::OnExcludePathsChanged)

.HintText(FText::FromString(TEXT("Exclude paths (comma separated)")))

]

]

+ SGridPanel::Slot(0, 3)

.Padding(2.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Run Mode")))

]

+ SGridPanel::Slot(1, 3)

.Padding(2.0f)

[

SNew(SComboBox<FString>)

.OptionsSource(&RunModes)

.OnGenerateWidget(this, &SOptimizerPanel::OnGenerateRunModeComboItem)

.OnSelectionChanged(this, &SOptimizerPanel::OnRunModeSelected)

.Content()

[

SNew(STextBlock)

.Text(this, &SOptimizerPanel::GetCurrentRunModeText)

]

]

]

// Main content area

+ SVerticalBox::Slot()

.FillHeight(1.0f)

.Padding(4.0f)

[

SNew(SScrollBox)

+ SScrollBox::Slot()

.Padding(4.0f)

[

SNew(SVerticalBox)

// Textures accordion

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(2.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Textures")))

]

// Meshes accordion

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(2.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Meshes")))

]

// Materials accordion

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(2.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Materials")))

]

// Levels accordion

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(2.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Levels")))

]

]

]

// Footer buttons

+ SVerticalBox::Slot()

.AutoHeight()

.Padding(4.0f)

[

SNew(SHorizontalBox)

+ SHorizontalBox::Slot()

.AutoWidth()

.HAlign(HAlign\_Left)

.Padding(4.0f)

[

SNew(SButton)

.Text(FText::FromString(TEXT("Run Audit")))

.OnClicked(this, &SOptimizerPanel::OnAuditClicked)

]

+ SHorizontalBox::Slot()

.AutoWidth()

.HAlign(HAlign\_Left)

.Padding(4.0f)

[

SNew(SButton)

.Text(FText::FromString(TEXT("Apply")))

.OnClicked(this, &SOptimizerPanel::OnApplyClicked)

]

+ SHorizontalBox::Slot()

.FillWidth(1.0f)

.HAlign(HAlign\_Right)

.Padding(4.0f)

[

SNew(STextBlock)

.Text(FText::FromString(TEXT("Last run: 0 assets changed")))

]

]

];

}

FReply SOptimizerPanel::OnAuditClicked()

{

// This would call the Python audit function

FString Phase = TEXT("audit");

TArray<FString> Categories;

if (IsTexturesChecked() == ECheckBoxState::Checked)

{

Categories.Add(TEXT("Textures"));

}

if (IsMeshesChecked() == ECheckBoxState::Checked)

{

Categories.Add(TEXT("Meshes"));

}

if (IsMaterialsChecked() == ECheckBoxState::Checked)

{

Categories.Add(TEXT("Materials"));

}

if (IsLevelsChecked() == ECheckBoxState::Checked)

{

Categories.Add(TEXT("Levels"));

}

UOptimizerRun\* OptimizerRun = NewObject<UOptimizerRun>();

OptimizerRun->Run(Phase, Categories);

FNotificationInfo Info(FText::FromString(TEXT("Audit started...")));

Info.bUseSuccessFailIcons = true;

Info.ExpireDuration = 3.0f;

FSlateNotificationManager::Get().AddNotification(Info);

return FReply::Handled();

}

FReply SOptimizerPanel::OnApplyClicked()

{

// This would call the Python apply function

FString Phase = TEXT("apply");

TArray<FString> Categories;

if (IsTexturesChecked() == ECheckBoxState::Checked)

{

Categories.Add(TEXT("Textures"));

}

if (IsMeshesChecked() == ECheckBoxState::Checked)

{

Categories.Add(TEXT("Meshes"));

}

if (IsMaterialsChecked() == ECheckBoxState::Checked)

{

Categories.Add(TEXT("Materials"));

}

if (IsLevelsChecked() == ECheckBoxState::Checked)

{

Categories.Add(TEXT("Levels"));

}

UOptimizerRun\* OptimizerRun = NewObject<UOptimizerRun>();

OptimizerRun->Run(Phase, Categories);

FNotificationInfo Info(FText::FromString(TEXT("Apply started...")));

Info.bUseSuccessFailIcons = true;

Info.ExpireDuration = 3.0f;

FSlateNotificationManager::Get().AddNotification(Info);

return FReply::Handled();

}

ECheckBoxState SOptimizerPanel::IsTexturesChecked() const

{

return (OptimizerSettings->CategoryMask & (uint8)EOptimizerCategory::Textures) ? ECheckBoxState::Checked : ECheckBoxState::Unchecked;

}

void SOptimizerPanel::OnTexturesChanged(ECheckBoxState NewState)

{

if (NewState == ECheckBoxState::Checked)

{

OptimizerSettings->CategoryMask |= (uint8)EOptimizerCategory::Textures;

}

else

{

OptimizerSettings->CategoryMask &= ~((uint8)EOptimizerCategory::Textures);

}

}

ECheckBoxState SOptimizerPanel::IsMeshesChecked() const

{

return (OptimizerSettings->CategoryMask & (uint8)EOptimizerCategory::Meshes) ? ECheckBoxState::Checked : ECheckBoxState::Unchecked;

}

void SOptimizerPanel::OnMeshesChanged(ECheckBoxState NewState)

{

if (NewState == ECheckBoxState::Checked)

{

OptimizerSettings->CategoryMask |= (uint8)EOptimizerCategory::Meshes;

}

else

{

OptimizerSettings->CategoryMask &= ~((uint8)EOptimizerCategory::Meshes);

}

}

ECheckBoxState SOptimizerPanel::IsMaterialsChecked() const

{

return (OptimizerSettings->CategoryMask & (uint8)EOptimizerCategory::Materials) ? ECheckBoxState::Checked : ECheckBoxState::Unchecked;

}

void SOptimizerPanel::OnMaterialsChanged(ECheckBoxState NewState)

{

if (NewState == ECheckBoxState::Checked)

{

OptimizerSettings->CategoryMask |= (uint8)EOptimizerCategory::Materials;

}

else

{

OptimizerSettings->CategoryMask &= ~((uint8)EOptimizerCategory::Materials);

}

}

ECheckBoxState SOptimizerPanel::IsLevelsChecked() const

{

return (OptimizerSettings->CategoryMask & (uint8)EOptimizerCategory::Levels) ? ECheckBoxState::Checked : ECheckBoxState::Unchecked;

}

void SOptimizerPanel::OnLevelsChanged(ECheckBoxState NewState)

{

if (NewState == ECheckBoxState::Checked)

{

OptimizerSettings->CategoryMask |= (uint8)EOptimizerCategory::Levels;

}

else

{

OptimizerSettings->CategoryMask &= ~((uint8)EOptimizerCategory::Levels);

}

}

ECheckBoxState SOptimizerPanel::IsUseSelectionChecked() const

{

return OptimizerSettings->bUseSelection ? ECheckBoxState::Checked : ECheckBoxState::Unchecked;

}

void SOptimizerPanel::OnUseSelectionChanged(ECheckBoxState NewState)

{

OptimizerSettings->bUseSelection = (NewState == ECheckBoxState::Checked);

}

FString SOptimizerPanel::GetIncludePaths() const

{

return OptimizerSettings->IncludePathsCsv;

}

void SOptimizerPanel::OnIncludePathsChanged(const FText& NewText)

{

OptimizerSettings->IncludePathsCsv = NewText.ToString();

}

FString SOptimizerPanel::GetExcludePaths() const

{

return OptimizerSettings->ExcludePathsCsv;

}

void SOptimizerPanel::OnExcludePathsChanged(const FText& NewText)

{

OptimizerSettings->ExcludePathsCsv = NewText.ToString();

}

TSharedRef<SWidget> SOptimizerPanel::OnGenerateProfileComboItem(FString Item) const

{

return SNew(STextBlock)

.Text(FText::FromString(Item));

}

void SOptimizerPanel::OnProfileSelected(TSharedPtr<FString> SelectedItem, ESelectInfo::Type SelectInfo)

{

if (SelectedItem.IsValid())

{

OptimizerSettings->TargetProfile = \*SelectedItem;

}

}

FText SOptimizerPanel::GetCurrentProfileText() const

{

return FText::FromString(OptimizerSettings->TargetProfile);

}

TSharedRef<SWidget> SOptimizerPanel::OnGenerateRunModeComboItem(FString Item) const

{

return SNew(STextBlock)

.Text(FText::FromString(Item));

}

void SOptimizerPanel::OnRunModeSelected(TSharedPtr<FString> SelectedItem, ESelectInfo::Type SelectInfo)

{

if (SelectedItem.IsValid())

{

// Currently not used but can be added for future enhancements

}

}

FText SOptimizerPanel::GetCurrentRunModeText() const

{

return FText::FromString(TEXT("Audit")); // Default to Audit mode

}

Testing Strategy

Unit Tests

Texture Analysis: Test texture compression, LOD settings, and virtual texture handling.

Mesh Analysis: Test LOD generation, Nanite toggling, and collision simplification.

Material Analysis: Test normal map conversions, static switch cleanup, and sampler consolidation.

Runtime Settings: Test scalability buckets, shadow maps, post-processing quality adjustments.

Integration Tests

End-to-end Optimization: Run full optimization cycles for all categories.

Python Bridge Stability: Ensure the bridge handles different Python scripts and data.

UI Responsiveness: Validate that UI elements update correctly during operations.

Safety Mechanisms: Test dry-run modes, max-changes caps, and revert snapshots.

Performance Benchmarks

Optimization Speed: Measure time taken for audit, recommend, apply, and verify phases.

Memory Usage: Monitor memory consumption during optimization processes.

Regression Testing: Ensure that optimizations do not introduce regressions in asset quality or performance.

Performance Optimization

Incremental Scans: Implement incremental scans to reduce processing time for unchanged assets.

Batch Processing: Use batch operations for large datasets to improve performance.

Threading and Asynchronous Loading: Leverage Unreal Engine's threading capabilities for parallel processing.

Safety Implementation

Error Handling: Robust error handling with proper logging for all critical operations.

Verification Systems: Implement verification systems to ensure correctness after optimizations.

Revert Mechanisms: Provide full revert snapshots and history tracking to undo changes if necessary.

Conclusion

This detailed implementation plan provides a comprehensive roadmap for developing the "MagicOptimizer" plugin. Each phase focuses on specific tasks, ensuring that all critical components are covered. By following this plan, you can build a robust and efficient asset optimization tool for Unreal Engine