07:SSL/TLS ファイルダウンロード　07ClientAssignment.cpp 完成コード

＜ヘッダー部分、関数等略＞

```
//-----------------------------------------------------------//
//レンダー処理
void ClientAssignment07::Render()
{
	// 基底クラスのレンダー呼び出し
	SceneBase::Render();
	// imgui
	Graphics& graphics = Graphics::Instance();
	float screenWidth = static_cast<float>(graphics.GetScreenWidth());
	ImGui::SetNextWindowPos( WIND_POS, ImGuiCond_Once);
	ImGui::SetNextWindowSize( WIND_SIZE, ImGuiCond_Once);
	if (ImGui::Begin("Download", nullptr, ImGuiWindowFlags_None))
	{
		if (ImGui::Button("Start"))
		{
			// TODO 07_01
			// ダウンロードファイルの指定
		//jpgの場合は次の行を<#if 1> fbxは<#if 0>でビルド
#if 1
			// ホスト名(ドメイン部分)
			std::string hostname = "comp.ecc.ac.jp";
			// パス
			std::string path = "/img/";
			// ファイル名
			std::string filename = "mvdumy.jpg";
#else
			// ホスト名(ドメイン部分)
			std::string hostname = "10.14.10.40";
			// ファイルパス
			std::string path = "/";
			// ファイル名
			std::string filename = "cube000.fbx";
#endif
			if (recvTh.joinable())recvTh.join();
			recvTh = std::thread(&ClientAssignment07::FileDownload,
```

```
this, hostname, path, filename);
                }
                if (ImGui::Button(u8"タイトルへ"))
                {
                        SceneManager::Instance().ChangeScene(new SceneTitle());
                }

        }
        ImGui::End();
}

void ClientAssignment07::FileDownload(const std::string& hostname, const std::string&
path, const std::string& filename)
{
        // TODO 07_02
        // OpenSSLの初期化
        OPENSSL_init_ssl(0, nullptr);

        WSADATA wsaData;
        SOCKET sock = INVALID_SOCKET;
        addrinfo hints = {}, * addrInfo = nullptr;

        try {
                // WSA初期化
                if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
                        throw std::runtime_error("WSA初期化失敗");
                }

                // 通信方式設定
                hints.ai_family = AF_INET;                              // IPv4設定
                hints.ai_socktype = SOCK_STREAM;          // TCP設定
                hints.ai_protocol = IPPROTO_TCP;

                // ドメインとポート番号からアドレス情報を取得する
                const std::string port = DOWNLOAD_PORT ;
                if (getaddrinfo(hostname.c_str(), port.c_str(), &hints,
```

```
&addrInfo) != 0) {
                        throw std::runtime_error("ドメインからアドレス取得に失敗しました");
                }

                // ソケット作成
                sock = socket(addrInfo->ai_family, addrInfo->ai_socktype, addrInfo->ai_protocol);
                if (sock == INVALID_SOCKET) {
                        throw std::runtime_error("ソケットの生成に失敗しました");
                }

                // サーバに接続
                if (connect(sock, addrInfo->ai_addr, static_cast<int>(addrInfo->ai_addrlen)) == SOCKET_ERROR) {
                        throw std::runtime_error("connectに失敗しました");
                }

                // TODO 07_03
                // SSLコンテキスト作成
                SSL_CTX* ctx = SSL_CTX_new(TLS_client_method());
                if (!ctx) {
                        throw std::runtime_error("SSL_CTXの生成に失敗しました");
                }

                // TODO 07_04
                // SSLオブジェクト作成
                SSL* ssl = SSL_new(ctx);
                if (!ssl) {
                        SSL_CTX_free(ctx);
                        throw std::runtime_error("SSLの生成に失敗しました");
                }

                // TODO 07_05
                // SSLオブジェクトにソケットを関連付ける
                if (SSL_set_fd(ssl, static_cast<int>(sock)) == 0) {
```

```
        SSL_free(ssl);
        SSL_CTX_free(ctx);
        throw std::runtime_error("ソケットとSSLの関連付けに失敗");
}

// TODO 07_06
// サーバに接続
if (SSL_connect(ssl) <= 0) {
        SSL_free(ssl);
        SSL_CTX_free(ctx);
        throw std::runtime_error("SSL接続に失敗しました");
}

// TODO 07_07
// HTTPリクエストを作成
char request[ REQUEST_BUF_SIZE ];
snprintf(request, sizeof(request),
        "GET %s%s HTTP/1.1¥r¥nHost: %s¥r¥nConnection:
        Close¥r¥n¥r¥n",
        path.c_str(), filename.c_str(), hostname.c_str());

// TODO 07_08
// リクエスト送信
if (SSL_write(ssl, request, static_cast<int>(strlen(request))) <= 0)
 {
        throw std::runtime_error("送信に失敗しました");
}

Logger::Print("サーバからのレスポンス¥n");
std::vector<char> data;
data.reserve( DATA_KEEP_SIZE ); // 初期サイズを確保
char buf[ BUFFER_SIZE ];
int size;


// TODO 07_09
```

```
        // データ受信ループ
        while ((size = SSL_read(ssl, buf, sizeof(buf))) > 0) {
                Logger::Print("%s",buf);
                data.insert(data.end(), buf, buf + size);
        }

        if (size < 0) {
                throw std::runtime_error("受信エラー");
        }

        // HTTPヘッダーの削除
        auto headerEnd = std::search(data.begin(), data.end(), "¥r¥n¥r¥n",
"¥r¥n¥r¥n" + 4);
        if (headerEnd != data.end()) {
                data.erase(data.begin(), headerEnd + 4);
        }

        // ファイル書き出し
        std::ofstream writingFile(filename, std::ios::binary);
        if (!writingFile) {
                throw std::runtime_error("ファイルのオープンに失敗しました
");
        }
        writingFile.write(data.data(), data.size());

        // TODO 07_10
        // リソース解放
        SSL_shutdown(ssl);
        SSL_free(ssl);
        SSL_CTX_free(ctx);

        closesocket(sock);
        WSACleanup();
    }
    catch (const std::exception& ex) {
        Logger::Print("エラー:%s¥n",ex.what());
```

07:SSL/TLS ファイルダウンロード　07ClientAssignment.cpp 完成コード

```
                if (sock != INVALID_SOCKET) closesocket(sock);
                WSACleanup();
        }
}
```