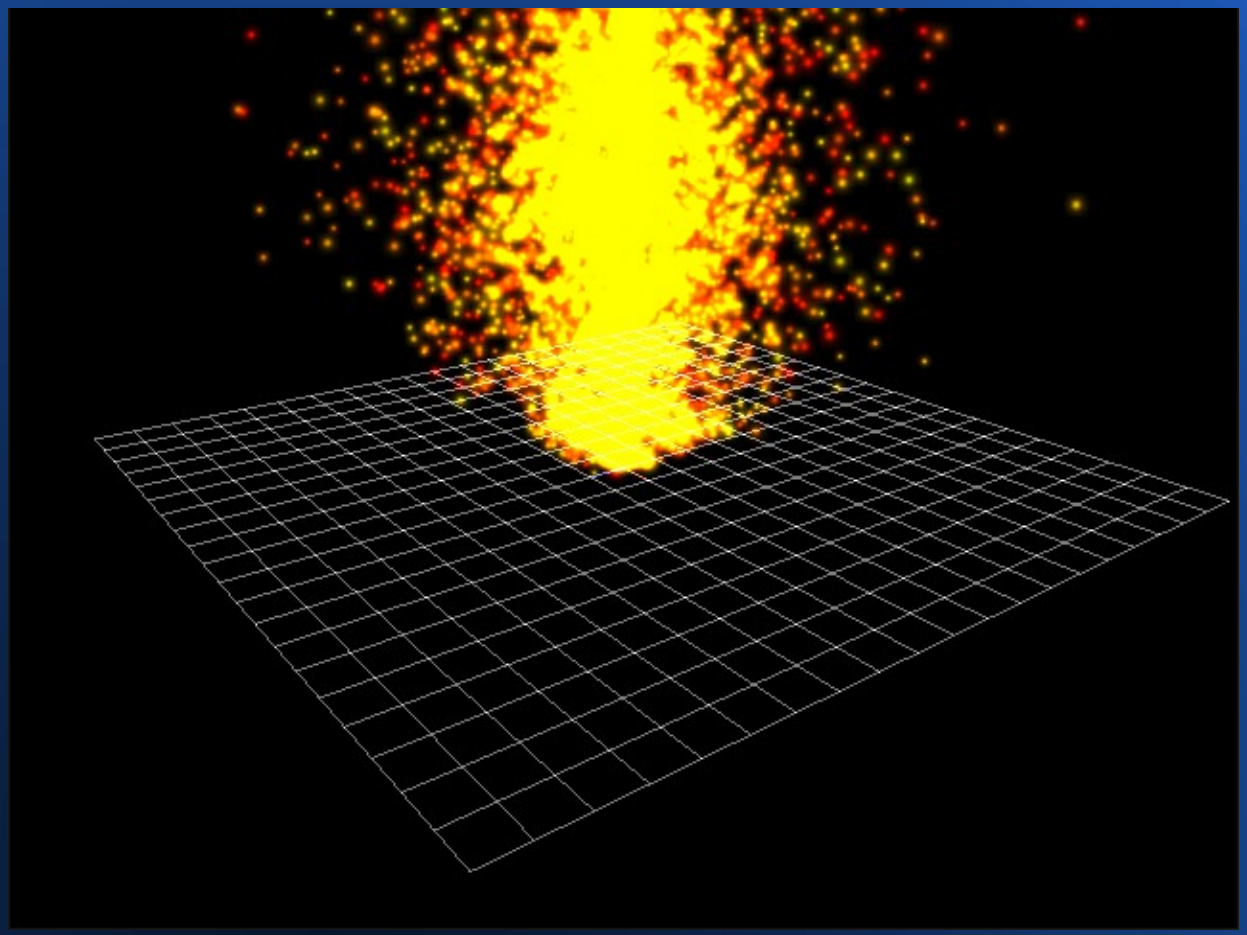
Seminarium dyplomowe

Wysokowydajny, wielowątkowy silnik modularnych efektów cząsteczkowych

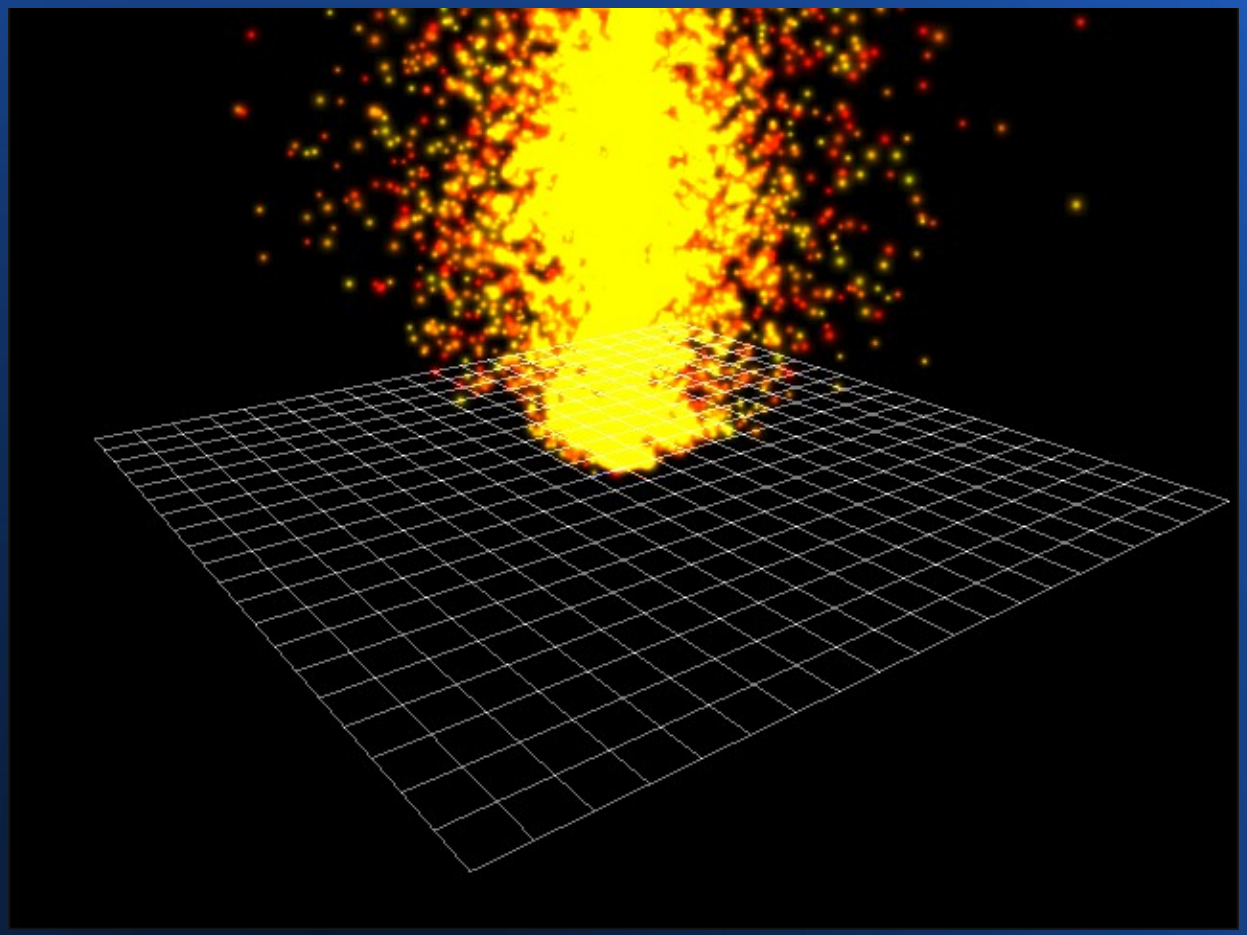
Student: Leszek Godlewski
Promotor: dr inż. Agnieszka Szczęsna
Konsultant: mgr inż. Jakub Stępień
Wydział AEI, Politechnika Śląska 2012

System cząstek

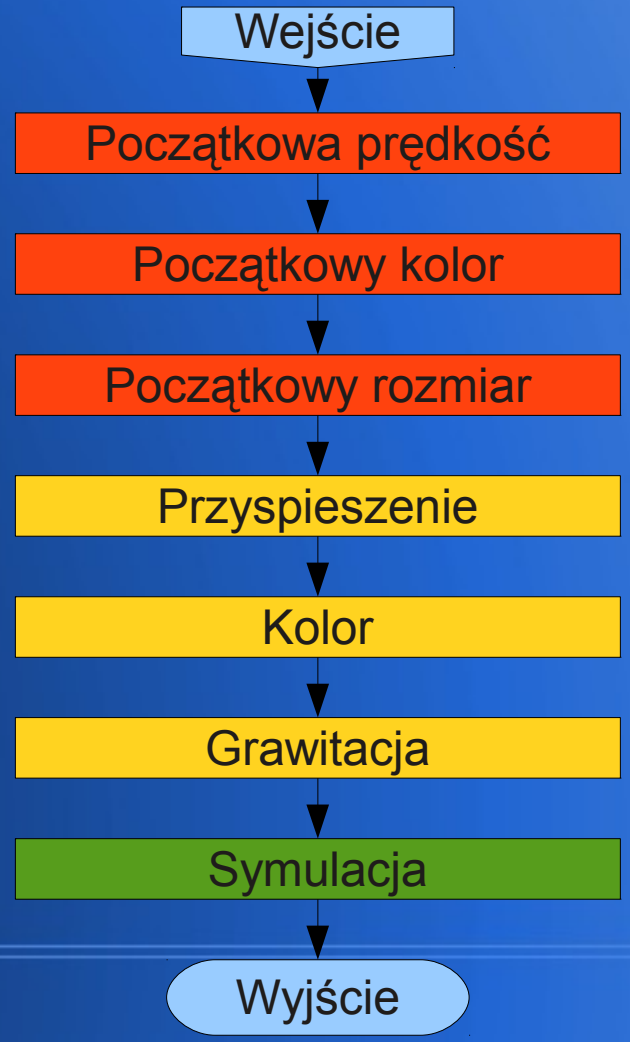


- Technika grafiki komputerowej służąca do symulacji zjawisk cząsteczkowych (płomienie, dymy, opady atmosferyczne, liście, smugi)
- Typowa implementacja:
  - Źródło (emiter)
  - Dwie fazy działania:
    - Faza symulacji
    - Faza renderingu
      - Typowa forma renderowania cząsteczki – oteksturowany billboard

Modularność



Analogia do łańcucha efektów DSP



Plan pracy inżynieskiej

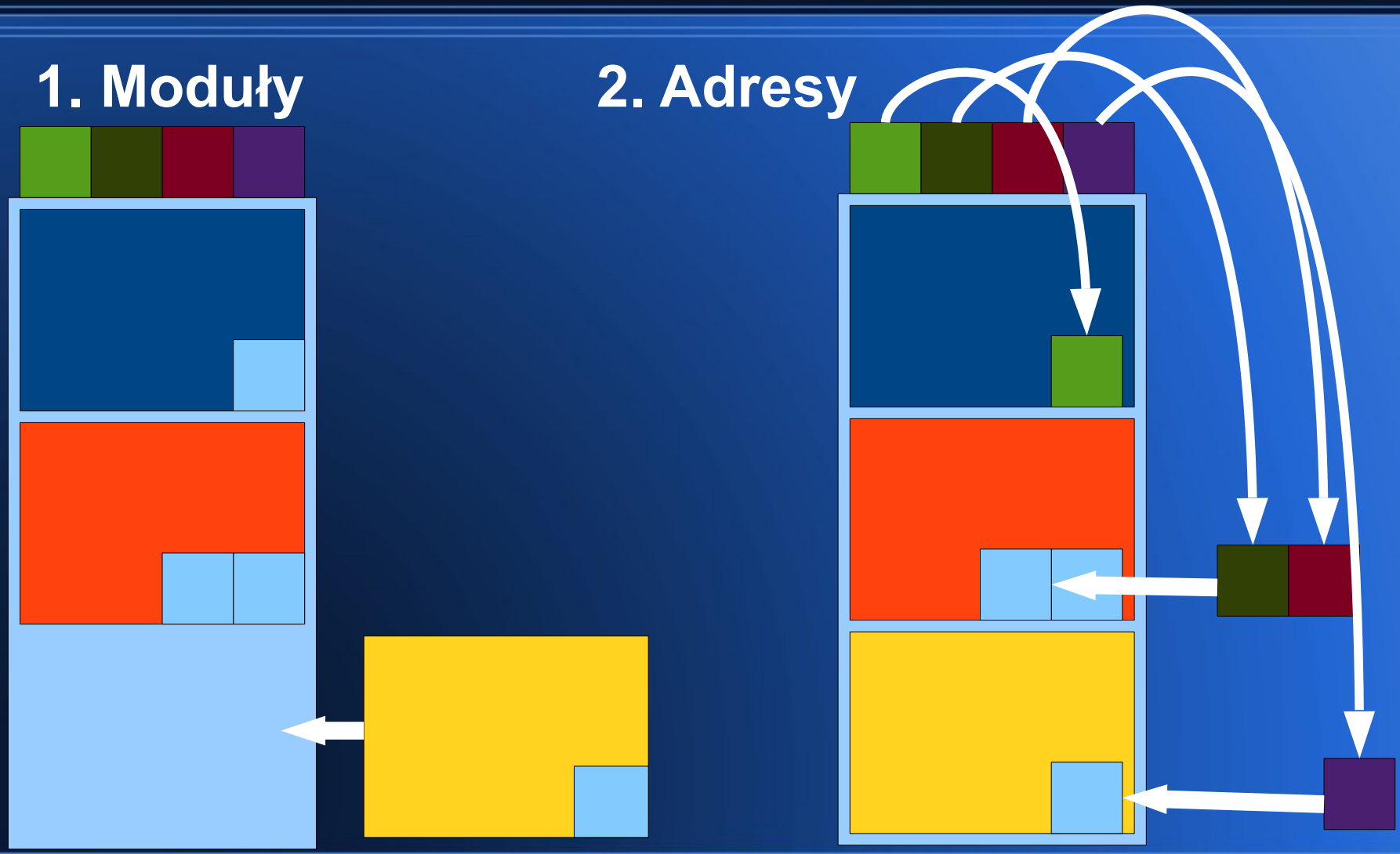
Plan minimum:

- Port silnika na architekturę x86-64
- Port „kleju” na platformę Windows
- Stworzenie graficznego edytora emiterów

Plan optymistyczny:

- Integracja np. z silnikiem OGRE 3D
- Wsparcie dla wielowątkowości
- Zastąpienie kompilatora generatorem kodu

Pierwotna architektura kompilatora



Problemy przy portowaniu kompilatora na x86-64

- Różnice w rozmiarach i offsetach struktur danych
- Rozgałęzianie kodu
- Konwencje wywołań
- Brak obsługi niektórych wariantów rozkazów w trybie 64-bit
- Czarę goryczy przelało wywoływanie funkcji za pośrednictwem RAX

Problemy przy portowaniu kompilatora na x86-64

```
%ifidn __BITS__,64
    push __ax
    mov  __ax, [0xDEADBEEF]
    call [__ax]
    pop  __ax
%else
    call [0xDEADBEEF]
%endif
```

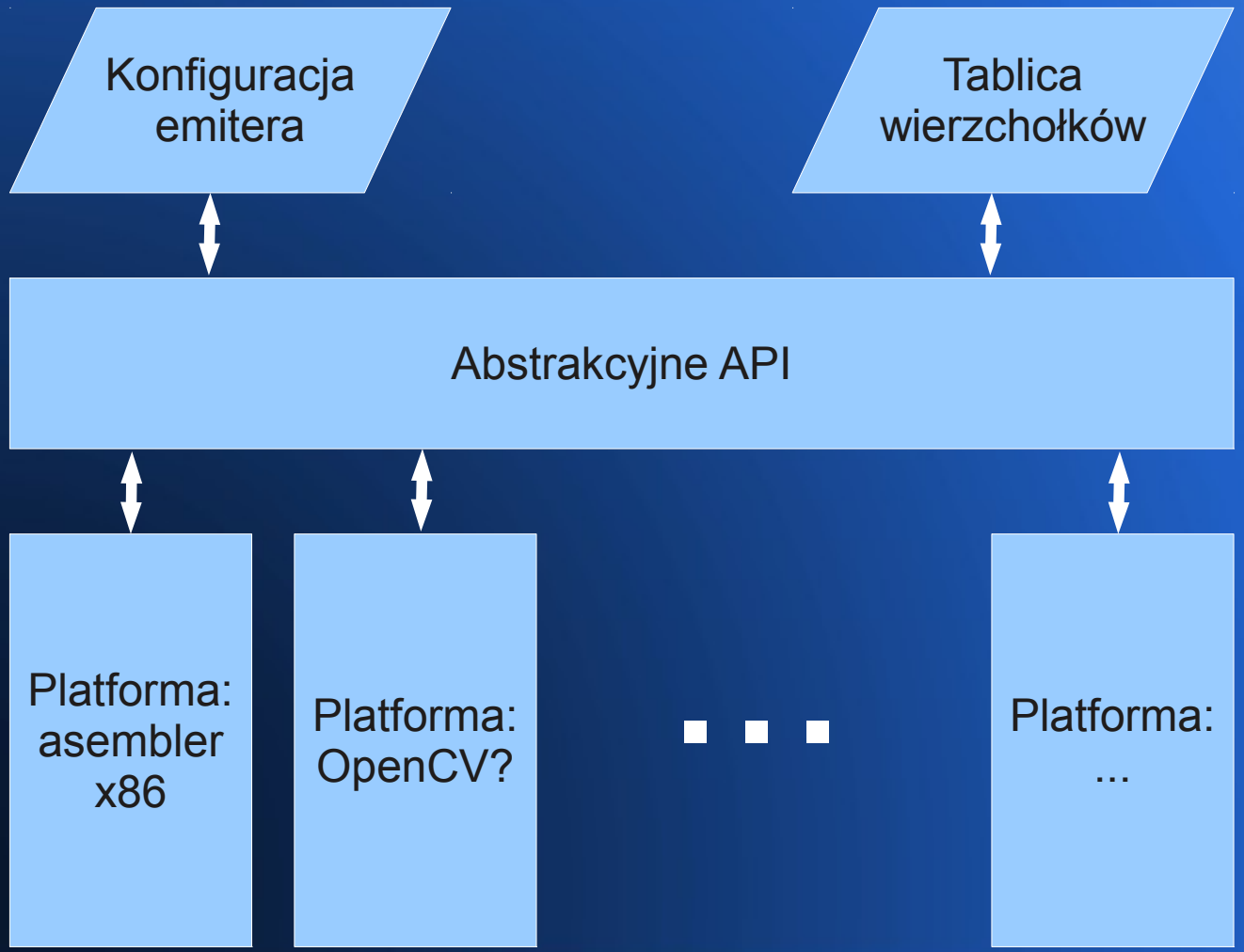
Problemy przy portowaniu kompilatora na x86-64

```
%ifidn __BITS__,64
    push __ax
    mov  __ax, [0xDEADBEEF]
    call [__ax]
    pop  __ax
%else
    call [0xDEADBEEF]
%endif
```

Zmiana planu

- Generator kodu – najpierw!
- Port na x86-64 i Windows
- Bardzo uproszczony edytor graficzny

Generator – architektura



Generator – architektura

- Retargetable
- Target asemblera x86
  - Generator w przenośnym C++
  - Kawałki kodu w asemblerze
  - Preprocesory w Pythonie
  - Kod uruchamiający – C++ ze wstawkami asemblera
  - Zależność od asemblera NASM

Stan projektu

- Generator: działa wszędzie
- Kod uruchomieniowy:
  - Linux x86: działa
  - Linux x86-64: nie działa
  - Windows x86: działa
  - Windows x86-64: nie działa
- Edytor: nierozpoczęty

Ciekawy problem #1: Procesy potomne

- Różne API
  - POSIX vs WinAPI
- Potoki
  - Przechwytywanie stdout, stderr
- Kod wyjścia procesu potomnego
  - Wykrywanie błędów

Ciekawy problem #1: Procesy potomne

Ogólny algorytm:

- Stworzenie potoków
- Uruchomienie procesu potomnego
- Zastąpenie urządzeń std* potomka potokami
- Oczekiwanie na zakończenie potomka
- Odczyt z potoków, odczyt kodu wyjścia i zwolnienie zasobów

Ciekawy problem #1: Procesy potomne

Więcej informacji:

- POSIX
  - How to capture stdin, stdout and stderr of child program!

    http://jineshkj.wordpress.com/2006/12/22/how-to-capture-stdin-stdout-and-stderr-of-child-program/
- Windows
  - Creating a Child Process with Redirected Input and Output

    http://msdn.microsoft.com/en-us/library/windows/desktop/ms682499(v=vs.85).aspx

Ciekawy problem #2: Preprocesory

- Potrzeba jednolitych definicje struktur w C++ i asemblerze
- Potrzeba wbudowania treści kawałków kodu asemblerowego generatora w kod C++
- Potrzeba wiedzy co do targetów wkompilowywanych w bibliotekę
- Rozwiązanie: skrypty w Pythonie!
  - Uruchamiane w procesie konstrukcji przed kompilacją

Ciekawy problem #2: Preprocesory

- GenerateConfig.py
  - Generuje plik nagłówkowy z #definicjami budowanych targetów
- GenerateAsmInclude.py
  - Parsuje nagłówek z API biblioteki i generuje analogiczny w asemblerze
- GenerateSnippetHeader.py
  - Generuje plik nagłówkowy z kawałków kodu asemblerowego do wbudowania w kod C++

Pytania

?

Dziękuję za uwagę

!