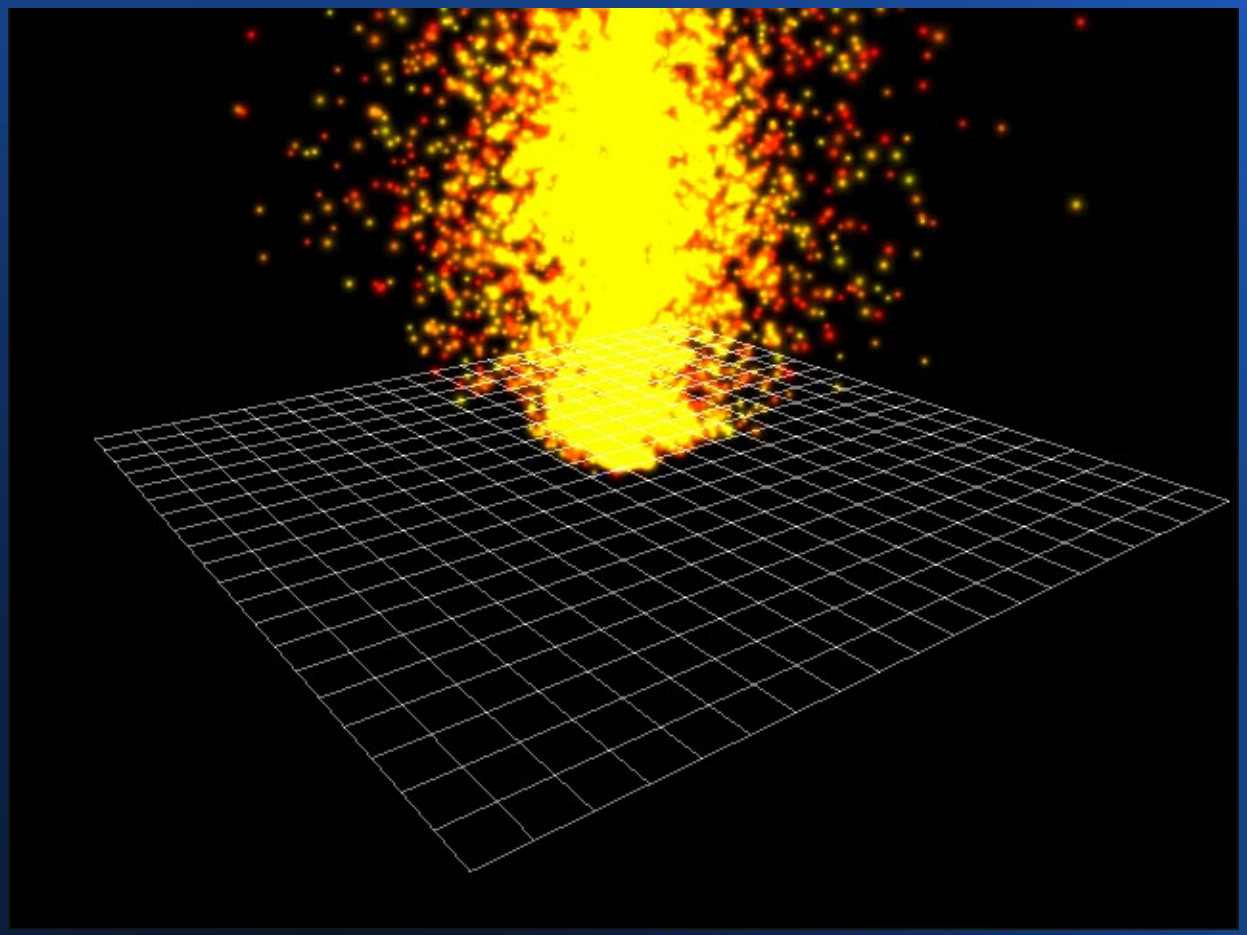
Seminarium dyplomowe

Wysokowydajny, wielowątkowy silnik modularnych efektów cząsteczkowych

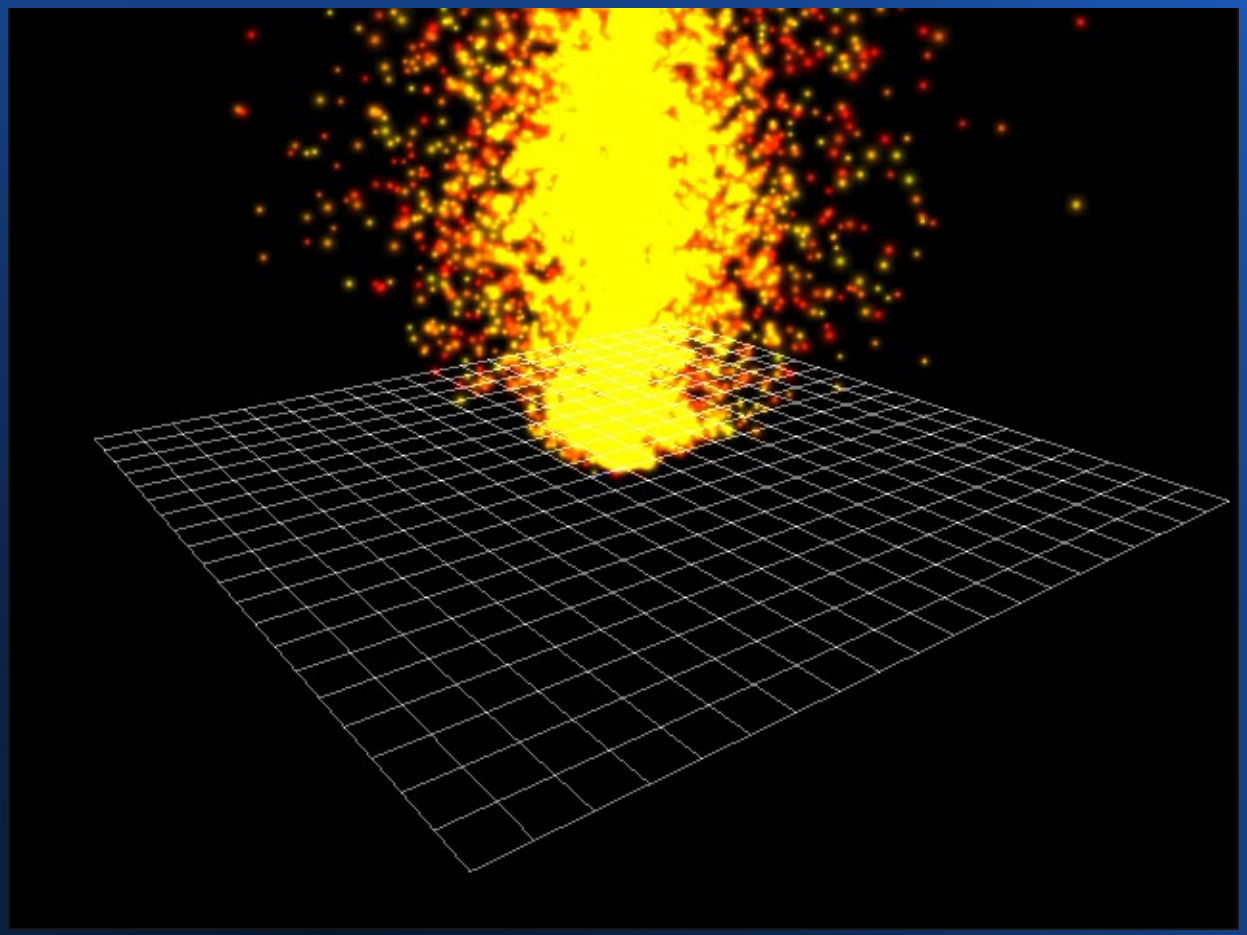
Student: Leszek Godlewski
Promotor: dr inż. Agnieszka Szczęsna
Konsultant: mgr inż. Jakub Stępień
Wydział AEI, Politechnika Śląska 2012

System cząstek

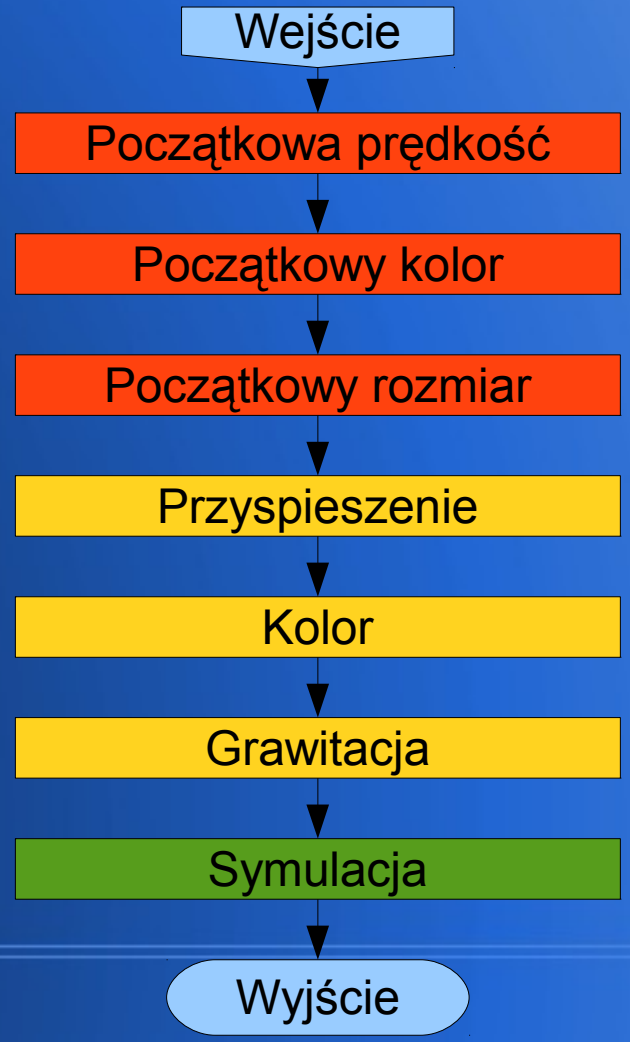


- Technika grafiki komputerowej służąca do symulacji zjawisk cząsteczkowych (płomienie, dymy, opady atmosferyczne, liście, smugi)
- Typowa implementacja:
  - Źródło (emiter)
  - Dwie fazy działania:
    - Faza symulacji
    - Faza renderingu
      - Typowa forma renderowania cząsteczki – oteksturowany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

Modularność



Analogia do łańcucha efektów DSP



Wydajność

Popularne i łatwe modele programowania

↓

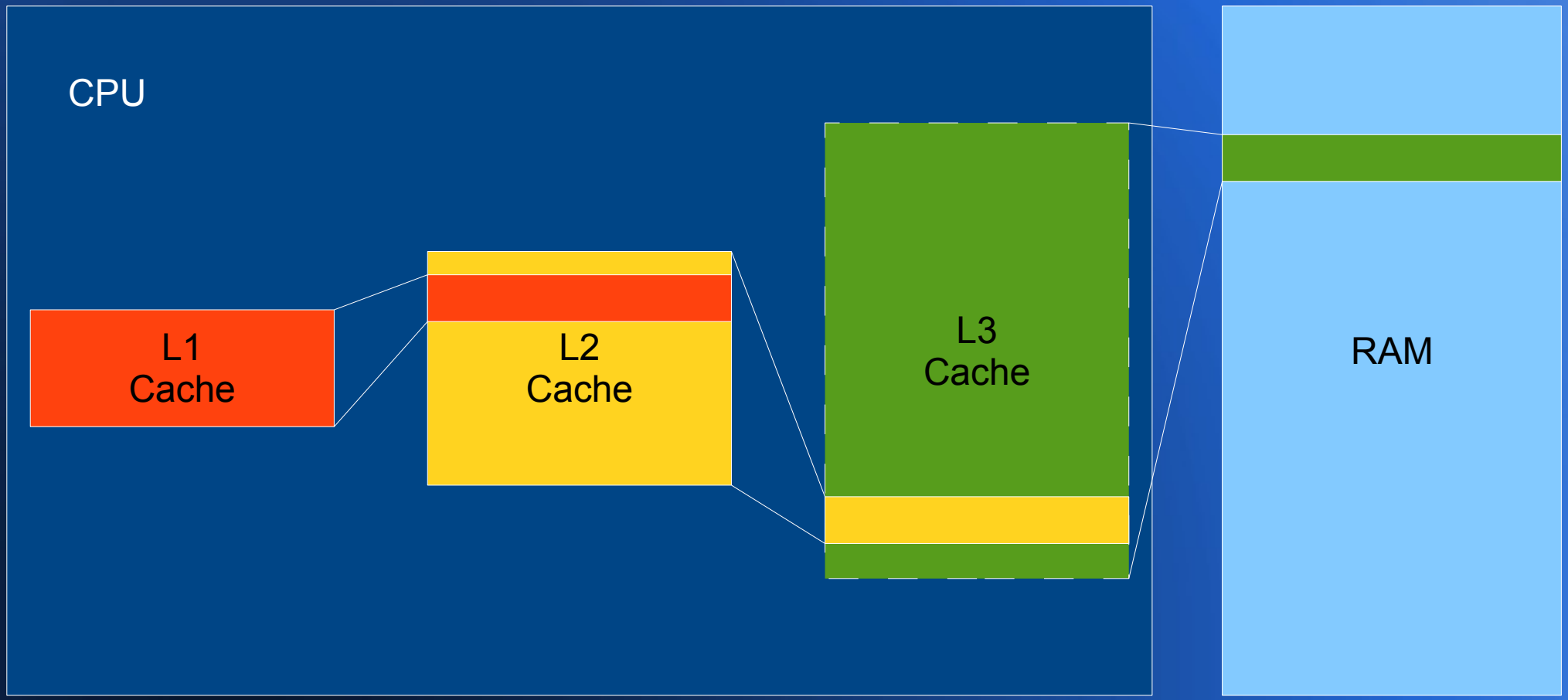
Rozproszenie i fragmentacja danych oraz kodu w pamięci

(np. narzut polimorfizmu i f-cji wirtualnych C++ – dereferencja obiektu → dereferencja vtable → dereferencja wskaźnika na funkcję)

↓

„Pudłowanie” pamięci podręcznej, błędy przewidywania rozgałęzień...

Wydajność

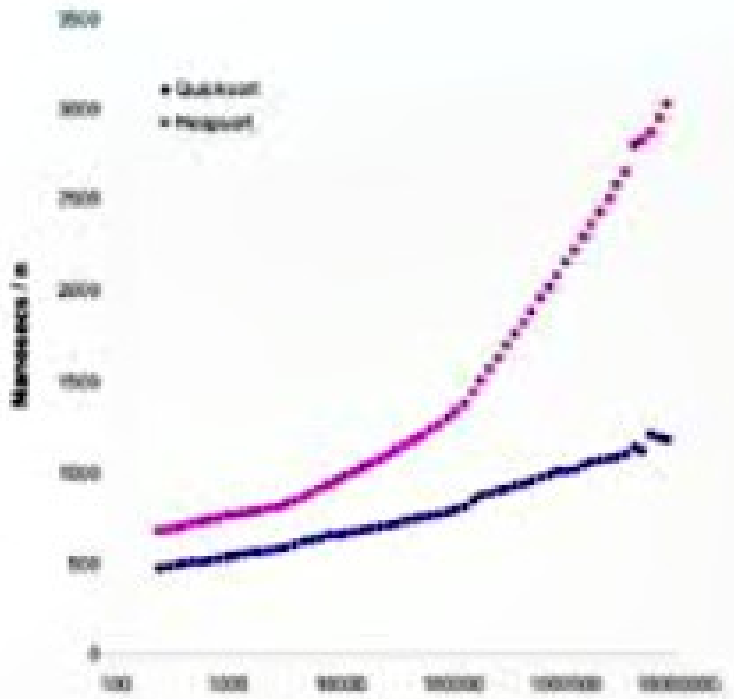


Wydajność

AVAYA

Analysis of Quicksort

- First approach
  - Measure real runs
  - Plot averages
    - of many runs
- Observation
  - Run times are
  - (ahem) interesting
- Alternative
  - Count key operations
  - Compares, swaps

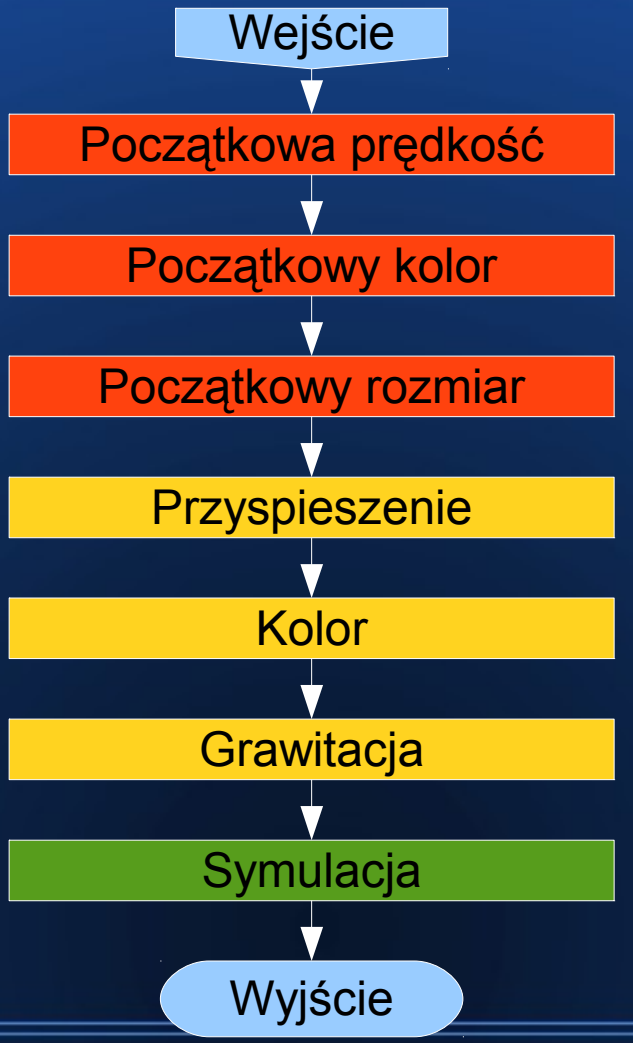


(źródło: Jon Bentley – Three Beautiful Quicksorts)

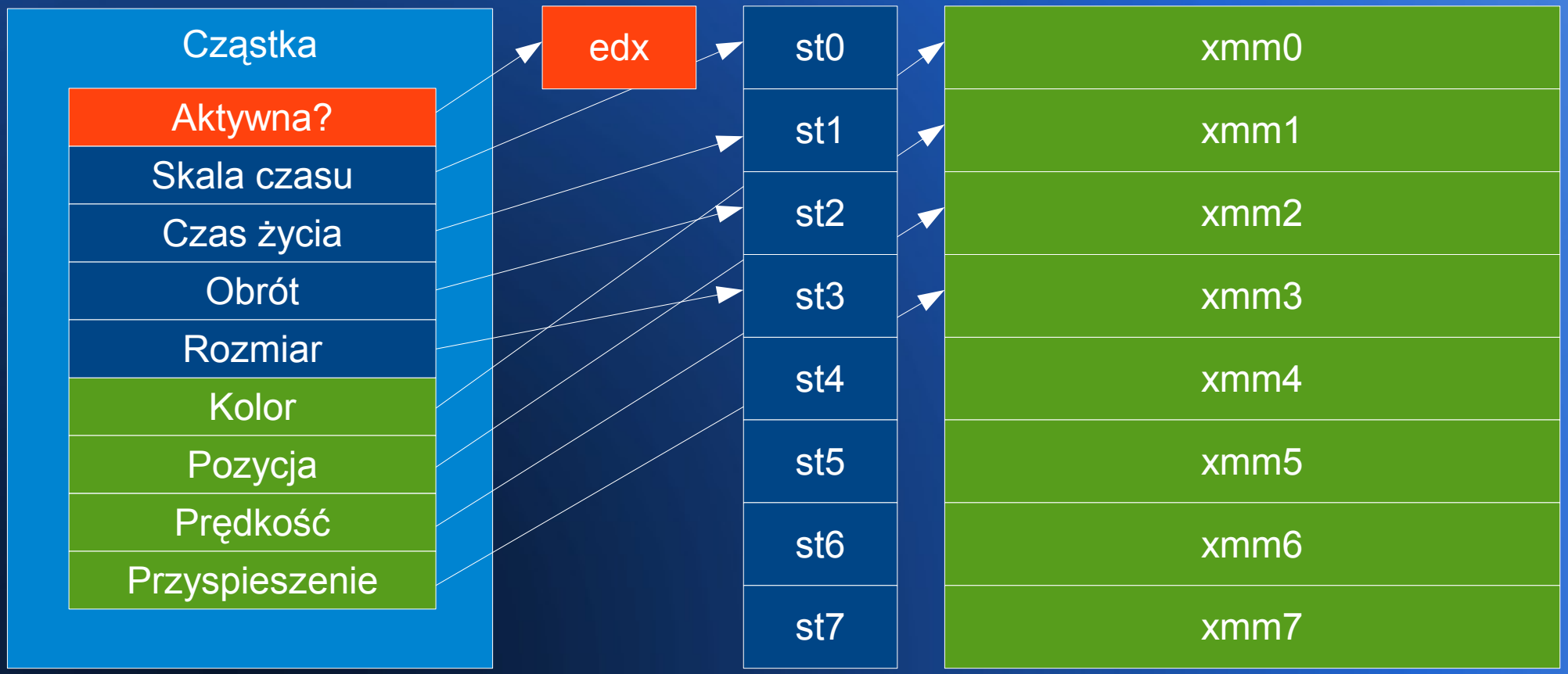
Zysk z programowania zorientowanego na dane

- Potraktowanie tablicy cząstek jako strumieni
- Równoległe przetwarzanie
- Unikanie skoków i dereferencji
  - Środek: „kompilacja” modułów

Kompilacja modułów



Konwencja wywołań



Już zaimplementowano

- Implementacja referencyjna silnika w C++
- Implementacja silnika w asemblerze x86-32 z „klejem” dla systemu Linux
  - Kompilator emiterów i pętla przetwarzania cząstek napisane w asemblerze; „klej” w C
  - Zysk: od 350% do 500% krótszy czas obliczeń
- Przykładowy emiter
- Graficzny podgląd przykładowego emitera
- „Bezgłowy” benchmark implementacji

Wady istniejącej implementacji

- Wsparcie tylko dla architektury x86-32 pod kontrolą systemu Linux
- Trudność w konfiguracji emiterów (brak graficznego edytora)
- Brak wsparcia dla wielowątkowości
- Trudny w utrzymaniu kod, zwłaszcza kompilatora emiterów
- Brak przykładu integracji z silnikiem graficznym

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

Wyzwania

- Asembler!
- Architektura x86-32 wpisana w fundamentalne założenia istniejącego kodu asemblerowego
- Konieczność umożliwienia integracji edytora z istniejącymi edytorami silników
- Dostęp do buforów zupełnie nieprzemyślany pod kątem wielowątkowości

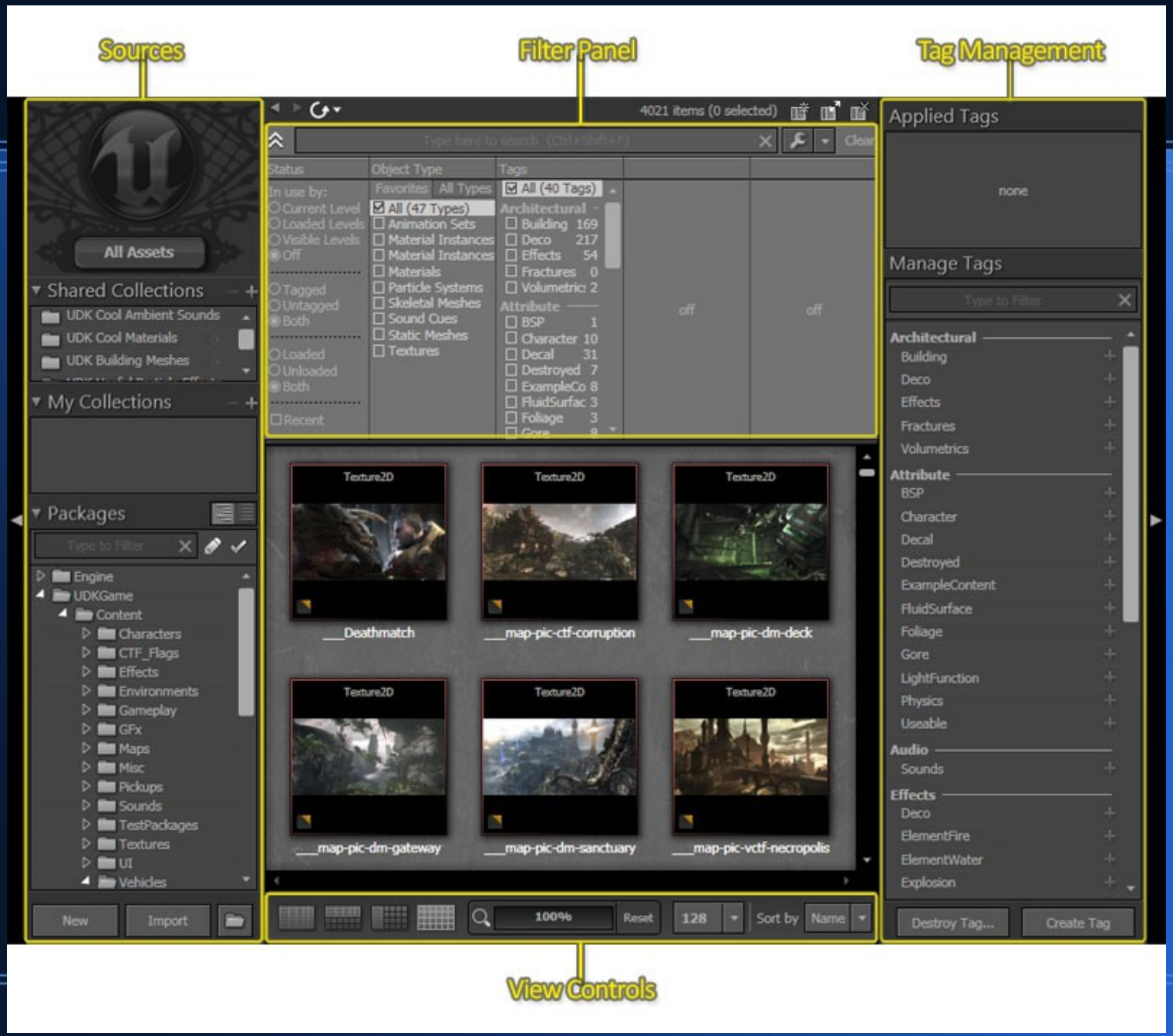
Port na architekturę x86-64 – wyzwania

- Inne nazwy rejestrów
- Inne rozmiary typów
- Inne konwencje wywołań
- Niektóre tryby adresowania z x86-32 nie działają w x86-64
  - Architektura x86-64 postuluje adresowanie względne wobec RIP
- Użycie FPU jest dezaprobowane na korzyść SSE

Port na architekturę x86-64 – pomysły

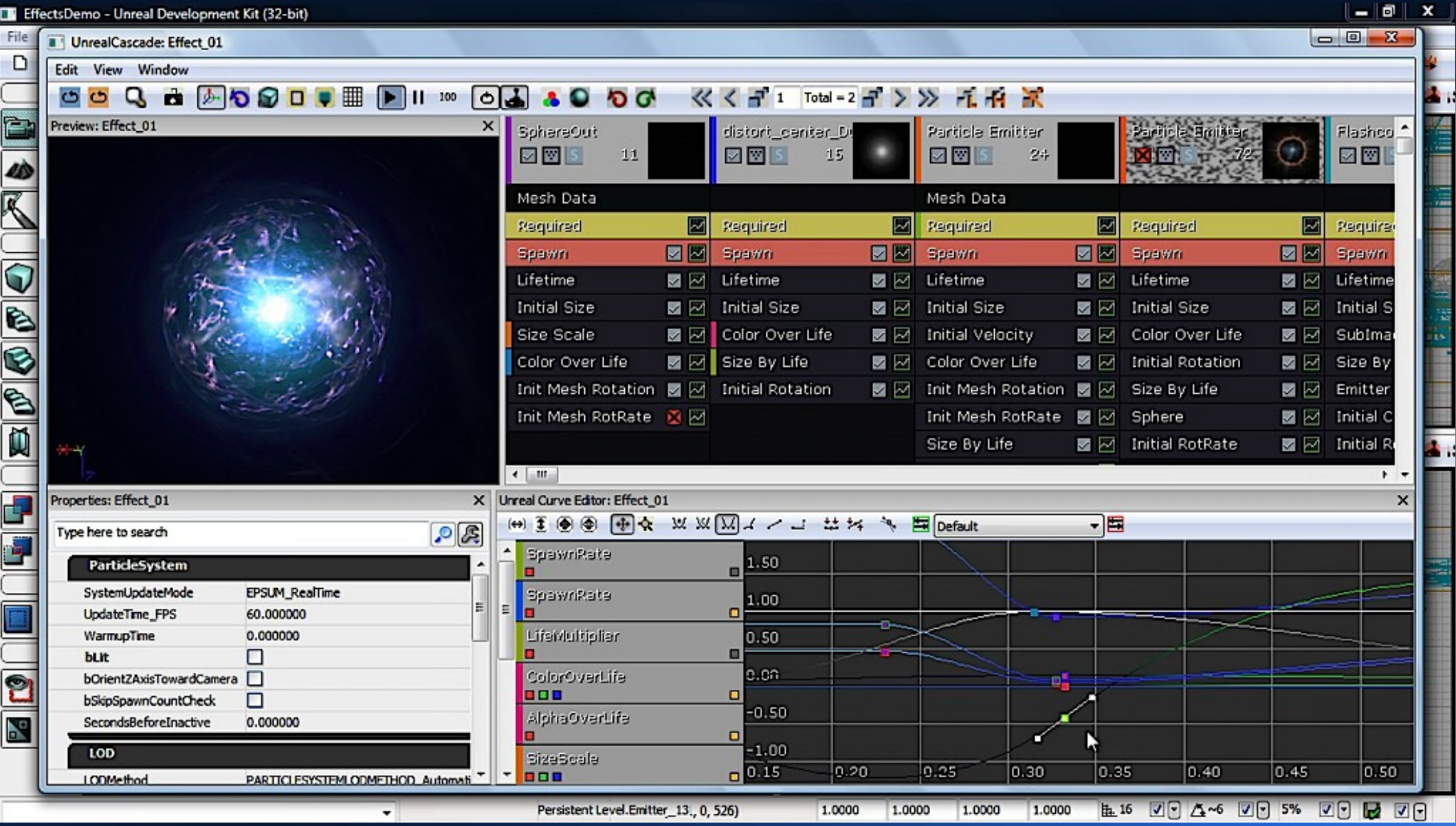
- NASMX – zestaw makr dla asemblera NASM, tworzących pewną warstwę abstrakcji
  - Rozwiązuje problemy nazw rejestrów oraz konwencji wywołań
  - Częściowo rozwiązuje problem typów
- Adresowanie względem IP na obu architekturach
- Użycie FPU jest „tylko” dezaprobowane :)
  - x86-64 oferuje więcej rejestrów XMM

„Integrowalny” edytor



(źródło: http://udn.epicgames.com/Three/ContentBrowserReference.html)

„Integrowalny” edytor



(źródło: http://www.joystiq.com/screenshots/unreal-development-kit-udk#/0)

„Integrowalny” edytor – wymagania

- Niedublowanie funkcjonalności oferowanej przez silnik-hosta!
  - Renderowanie okienka podglądu z użyciem silnika-hosta
    - Właściwy potok renderujący!
  - Pozostałe dane (materiały → tekstury + shadery)
  - Możliwość abstrakcji dostępu do danych
- Niezależność od toolkitu UI/pompy komunikatów
- Proste, zwięzłe API

„Integrowalny” edytor – pomysły

- Dwa elementy – klient i serwer – komunikujące się przez mechanizmy IPC
  - Mały, „bezgłowy” klient jako wtyczka do silnika-hosta – tylko wymiana danych
  - Serwer jako osobny proces - GUI i cała logika
- Serwer przekazuje klientowi (a ten – silnikowi-hostowi) natywny uchwyt okna podglądu
- Referencja do materiału (shader + tekstury) przechowywana jako łańcuch znaków

Generator kodu

Podejście obecne: „kompilacja” kodu emitera z skompilowanego kodu maszynowego, z uzupełnianiem adresów

- Błędogenność, trudność w utrzymaniu
- Absolutnie nieprzenośne!

Generator kodu

Podejście proponowane: generowanie kodu i integracja z asemblerem

- Przejście na język wyższego poziomu
- Uproszczenie architektury i kodu biblioteki
- Większa niezawodność i łatwość utrzymania
- Stworzenie potencjału do nowych zastosowań (np. demoscena)
- Przenośność!

Pytania

?

Dziękuję za uwagę

!