Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

Bartosz Bazyluk

GAME LOOP

Podstawowy element silnika gry komputerowej

Programowanie Gier Komputerowych, Informatyka S1, III Rok

Wydział
Informatyki

INFORMACJE ORGANIZACYJNE:

- **Prezentacje z wykładów, materiały, informacje** można znaleźć na stronie internetowej:
  - **http://bazyluk.net/dydaktyka**
- Warunkiem **zaliczenia przedmiotu** jest zrealizowanie zadania projektowego: *gry* lub *silnika gry*.
  - **Wspólna ocena** za projekt, laboratoria i cały przedmiot
  - Dopuszczalna jest **uczciwa współpraca w grupach** lub **praca indywidualna**
- **Wykłady** będą dotyczyły zagadnień związanych z **tworzeniem od podstaw silnika gry komputerowej**

TREŚĆ WYKŁADÓW:

Jest to lista **planowana**, może jeszcze ulec zmianie zależnie od Państwa potrzeb, możliwości czasowych i innych czynników.

- Game loop
  - Sposoby realizacji pętli głównej
- Współczesny OpenGL
  - *OpenGL 3.2+*, macierze przekształceń, *VBO*, *IBO*, *shadery*
- Organizacja sceny 3D
  - Hierarchia obiektów, architektura kodu
- Maszyna stanów
  - Sposób realizacji ciągłości przebiegu interakcji z grą
- Podział przestrzeni świata gry
  - Siatki, drzewa wspomagające przeszukiwanie świata
- Symulacja zjawisk kinetycznych w świecie gry
  - Użycie silnika fizycznego, fizyka brył sztywnych i miękkich
- Optymalizacja wydajności systemu renderującego
  - Techniki poprawy wydajności, użycie *profilera*

LITERATURA:

Dla osób, które zechcą zgłębić tematykę poruszaną w ramach przedmiotu.

- Literatura podstawowa:
  - Mike McShaffry, David Graham. ***"Game Coding Complete, Fourth Edition"***. Course Technology, 2012.
  - Jason Gregory. ***"Game Engine Architecture"***. AK Peters, 2009.
  - Tomas Akenine-Moller, Eric Haines, Naty Hoffman. ***"Real-Time Rendering, Third Edition"***. AK Peters, 2008.
- Literatura uzupełniająca:
  - Adam Lake. ***"Game Programming Gems 8"***. Course Technology, 2011.
  - Hubert Nguyen. ***"GPU Gems 3"***. Addison-Wesley, 2008.
  - Eric Lengyel. ***"Mathematics for 3D Game Programming and Computer Graphics, Third Edition"***. Course Technology, 2012.
  - Mat Buckland. ***"Programming Game AI by Example"***. Worldwide Publishing, 2005.

PLAN WYKŁADU:

- *Game loop* – **co to takiego**?
  - Elementy składowe: **aktualizacja stanu**, **renderowanie**
  - Pojęcia: **synchronizacja pionowa**, **FPS, frame time**
- **Biblioteki** narzędziowe dla OpenGL
  - GLUT, FreeGLUT, EGL, GLFW, etc.
- **Implementacja** game loop – różne metody:
  - Podejście "klasyczne"
  - FPS zależne od stałej szybkości gry
  - Szybkość gry zależna od FPS
  - Stała szybkość gry i max. FPS
  - Stała szybkość gry i max. FPS z interpolacją
  - Podejście **wielowątkowe**
    - interpolacja, predykcja, synchronizacja wątków

WPROWADZENIE:
Game loop

Uproszczona różnica w najbardziej **zasadniczej kwestii** związanej z działaniem programów o różnym charakterze.

ZASADA DZIAŁANIA PROGRAMU

Aplikacja użytkowa

- **Reaguje** na akcję użytkownika
- W pozostałym czasie **nie robi nic**

- **Event loop**

Gra komputerowa

- **Nieustannie** aktualizuje świat gry
- **Uwzględnia** akcję użytkownika

- **Game loop**

WPROWADZENIE:

Game loop

- **Nieskończona** pętla
- Warunkiem **stopu** jest wystąpienie **żądania** zakończenia działania programu
- Postać **podstawowej** wersji pętli:

```
while (użytkownik nie chce wyjść) {
    sprawdź urządzenia wejścia;
    wykonaj zadania AI;
    przemieść przeciwników;
    sprawdź kolizje;
    renderuj świat;
    odtwórz dźwięki;
}
```

WPROWADZENIE:
Game loop

Te pojęcia są **kluczowe** dla zrozumienia dalszej części wykładu!

- W dalszych rozważaniach przyjmiemy **uproszczenie**, że podstawowy game loop ma postać:

```
while (użytkownik nie chce wyjść) {
    aktualizuj stan gry;
    renderuj świat gry;
}
```

- Zatem **zdefiniujmy** następujące określenia:
  - **Aktualizacja** stanu gry
    - Wszystkie operacje wpływające na stan świata, wynik rozgrywki, pozycję obiektów, obsługa wejścia i zdarzeń, ...
  - **Renderowanie** świata gry
    - Wybór obiektów do renderowania, wyznaczenie ich macierzy przekształceń, zlecenie renderowania karcie graficznej, zamiana buforów, ...

WPROWADZENIE:
Synchronizacja pionowa

Zakładamy używanie techniki **podwójnego buforowania** (ang. *double buffering*).

***Front** buffer* jest buforem, który jest **wyświetlany**.

***Back** buffer* jest buforem, który **aktualizuje** karta.

W momencie zakończenia aktualizacji bufory są **zamieniane** miejscami (ang. *buffer swap*).

Źródła zdjęć:
Tech'n'Gear.com
PC Arena Hungary
The SEO Millionaire

- Obraz wyświetlany na monitorze odpowiada **aktualnej zawartości** *front buffera*.
- **Zawartość** *front buffera* jest:
  - **generowana** przez układ graficzny
  - **konsumowana** przez układ wysyłający obraz do monitora
- Oba procesy są **niezależne**.

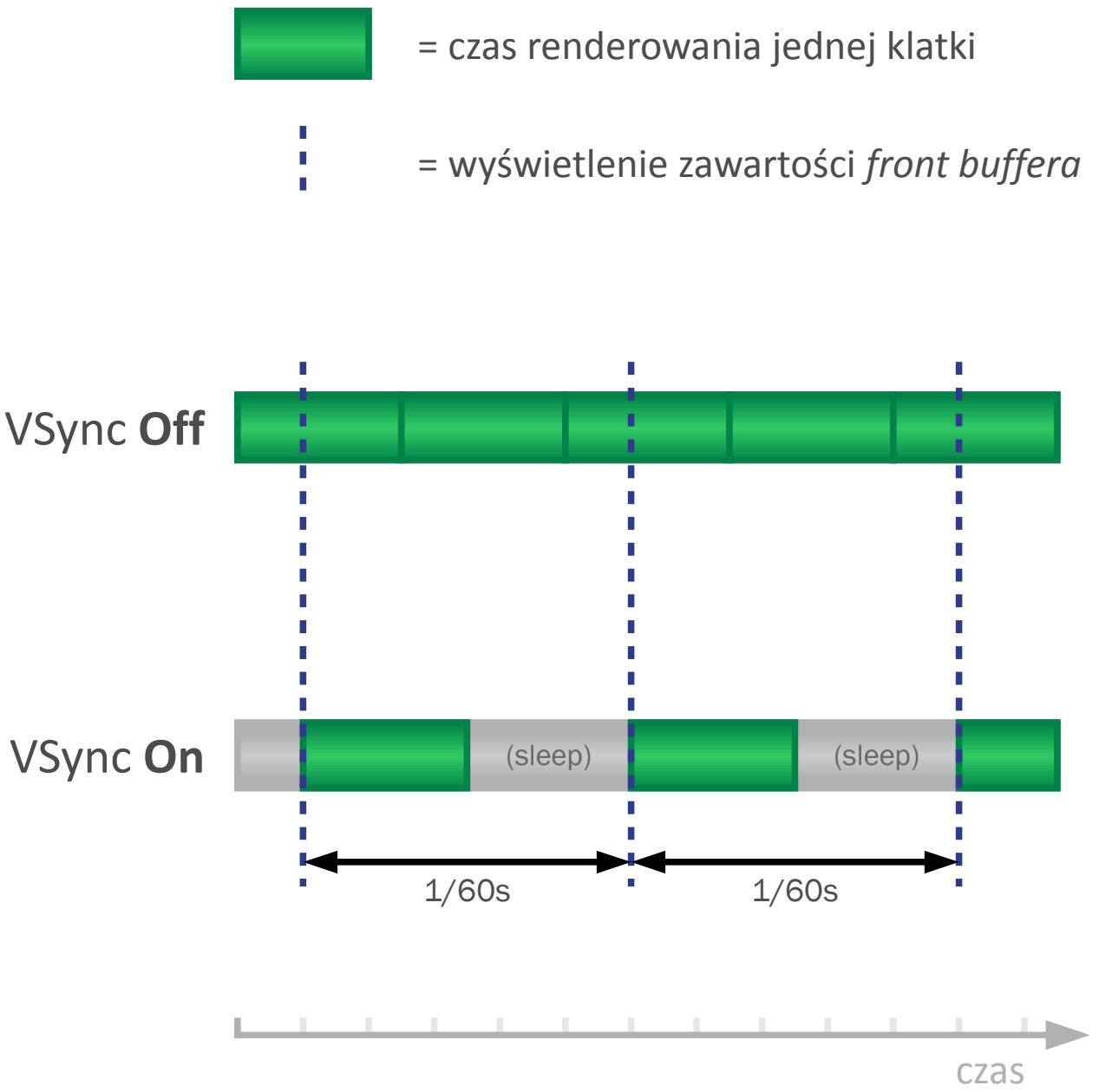
Aktualizacja świata
Żądanie renderowania

Renderowanie

Wyświetlanie

WPROWADZENIE:
Synchronizacja pionowa

Przebieg czasowy
renderowania klatek
w przypadku wyłączonej
i włączonej synchronizacji
pionowej.



WPROWADZENIE:
Synchronizacja pionowa

Porównanie kolejnych klatek wyświetlonych **na monitorze**.

Brak synchronizacji i podwójnego buforowania powoduje powstanie efektu ***tearing***: wyświetlony obraz składa się z kilku klatek.

VSync **On**
With **double-buffering**

VSync **Off**
No **double-buffering**

WPROWADZENIE:
Pomiar wydajności działania gry

Wpływ na wydajność potoku renderującego ma bardzo **wiele czynników**.

Renderowanie 10.000 wielokątów **nie jest** dwa razy szybsze niż renderowanie 20.000.

Więcej na ten temat:
http://www.opengl.org/wiki/Performance
http://www.mvps.org/directx/articles/fps_versus_frame_time.htm

- **Cząstkowy** pomiar czasu (np. rysowania jednego obiektu) jest **niemiarodajny**
  - kolejkowanie, buforowanie, etc. w karcie graficznej
- Wartościowy jest pomiar czasu **pomiędzy kolejnymi** momentami **zamiany buforów** (*buffer swap*)
  - niewiele mówi przy włączonej synchronizacji pionowej
- Liczba **klatek na sekundę** (ang. *frames per second*, ***FPS***)
  - miara **nieliniowa**
    - Spadek liczby FPS **z 900 do 450** jest tym samym, co **z 60 do 56.25**!
- Czas klatki (ang. *frame time*)
  - miara **liniowa**
- Mierzymy **zawsze z wyłączoną** synchronizacją pionową!

BIBLIOTEKI:
GL Utility Toolkit (GLUT)

Ostatnia wersja pochodzi z...
1996 roku!

Nie zalecana.

Witryna projektu:
http://www.opengl.org/resources/libraries/glut/

- Pozwala **szybko** przejść do sedna
- **Łatwo** się nauczyć
  - Pomija się przy okazji wiele istotnych kwestii
- Odpowiedni do:
  - Szybkiego **prototypowania**
  - Prostych **demonstracji**
- Problemy:
  - Niekonfigurowalna, narzucona implementacja **game loop**
  - Zamknięcie okna równoznaczne z **zakończeniem** programu
  - **Nieprzystosowany** do nowego OpenGL
- Nie jest open source => **nie ma dalszego rozwoju**

BIBLIOTEKI:
freeGLUT

Autorem pierwszych wersji jest **Polak**, Paweł Olszta.

Witryna projektu:
http://sourceforge.net/projects/freeglut/

- **Open-source**'owy klon GLUTa
- **Kompatybilny** z API GLUT
  - Zazwyczaj wystarczy podmienić DLL-kę
- Pozwala na:
  - tworzenie **własnej implementacji** game loop
  - wybór **profilu kontekstu** OpenGL
- Ciężki, cierpi z powodu przywiązania do API GLUT

BIBLIOTEKI:
GLFW

Ciągle rozwijana,
lekka alternatywa.

Będzie wykorzystywana
w dalszych przykładach.

Witryna projektu:
http://www.glfw.org/

- **Open-source**'owa alternatywa
- Ma za zadanie dostarczać tylko **podstawowej funkcjonalności**, zostawiając dużą dowolność
- **Lekka** i **prosta** w użyciu
- Możliwa **własna implementacja** game loop
- **Pełna kontrola** nad istotnymi elementami
- Wsparcie dla **wielowątkowości**

BIBLIOTEKI:
EGL

Najczęściej można się spotkać przy programowaniu na Androida

- API rozwijane przez **Khronos Group**
- To **nie jest biblioteka**, implementację dostarczają twórcy sprzętu/systemu/etc.
- Najpopularniejsze podejście przy programowaniu na **Androida**
- Składania i podejście zbliżone do samego **OpenGL**

Witryna projektu:
•http://www.khronos.org/egl/

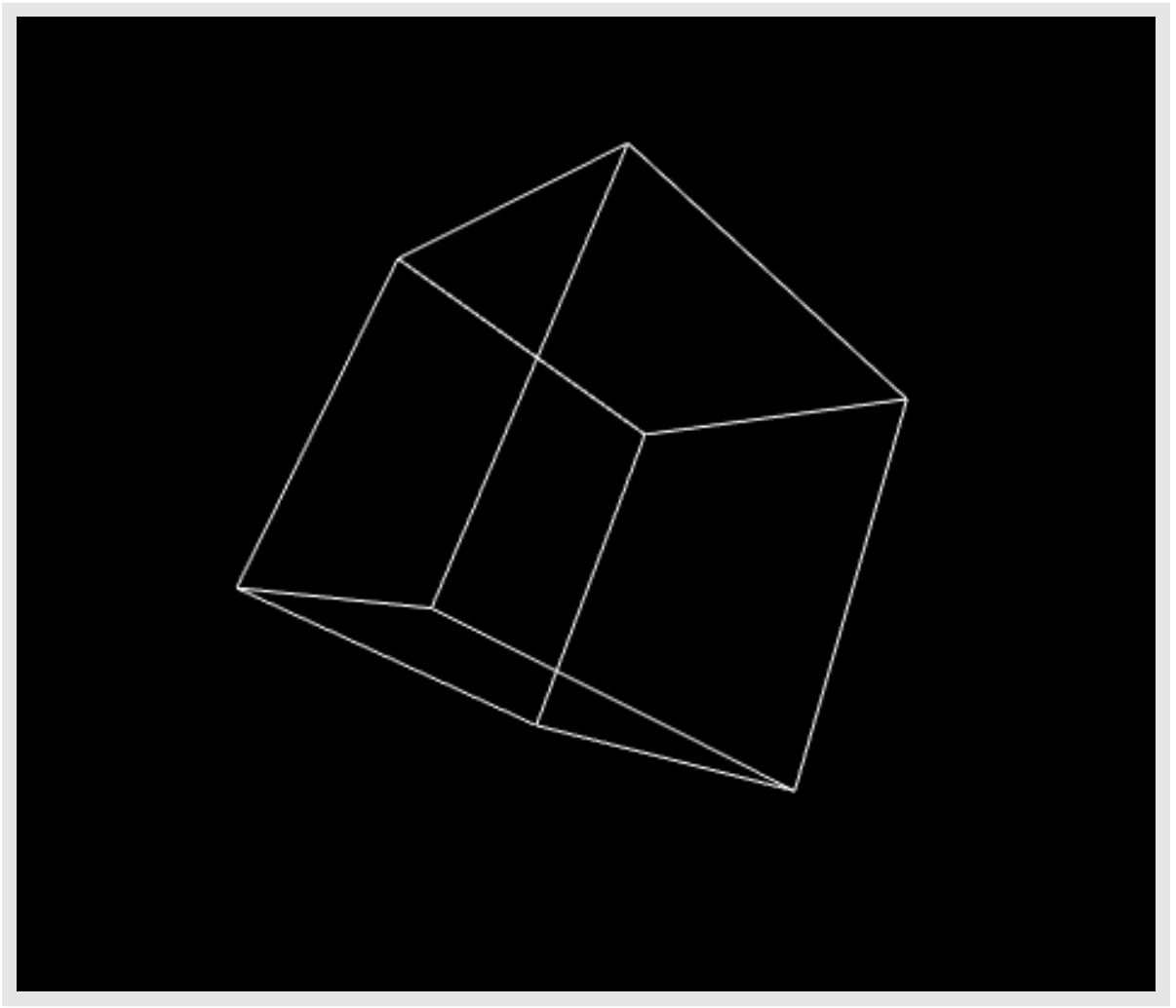
BIBLIOTEKI:
Inne przykłady

- Simple DirectMedia Layer (**SDL**)
  - http://www.libsdl.org
- Simple and Fast Multimedia Library (**SFML**)
  - http://www.sfml-dev.org
- **OpenGLUT**
  - http://openglut.sourceforge.org
- **Bez biblioteki**, bezpośrednie odwołania do systemu

IMPLEMENTACJA:
Wprowadzenie

Za przykład do analizy posłuży nam prosty, **obracający się sześcian**.

Kod przykładu jest **uproszczony** i celowo korzysta z archaicznego immediate mode.



Kody źródłowe dostępne na:
http://bazyluk.net/dydaktyka

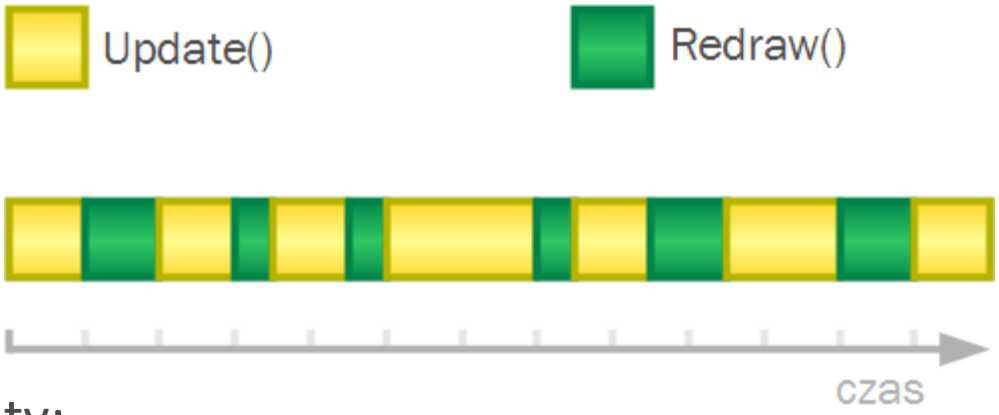
IMPLEMENTACJA:
Podstawowy game loop

Aktualizacja występuje ze **stałym krokiem**.

Absolutnie **nie jest** to podejście zalecane!

- Najprostsza forma:

```
while (użytkownik nie chce wyjść) {
    aktualizuj stan gry;
    renderuj świat gry;
}
```



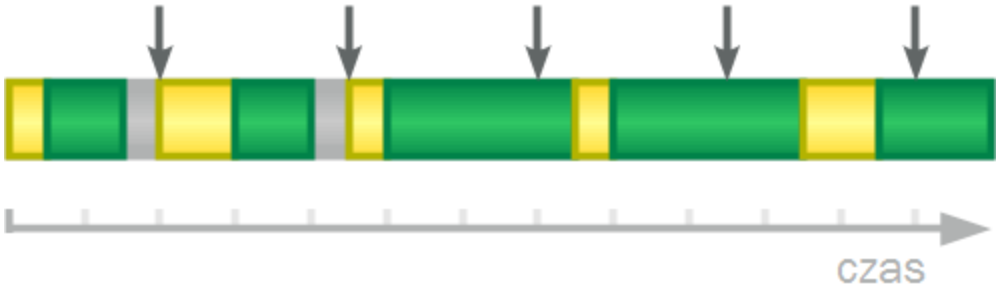
- Zalety:
  - Prostota (?)
- Problemy:
  - Szybkość rozgrywki zależna **od wydajności sprzętu**
  - Szybkość rozgrywki może się **zmieniać** w trakcie gry

IMPLEMENTACJA:

Stała szybkość gry

Podejście może być korzystne w przypadku gier na **urządzenia mobilne**. Może bowiem redukować zużycie energii.

- Określamy, **co ile czasu** ma wykonać się aktualizacja stanu gry.
- Każda aktualizacja stanu **zawsze** oznacza następujące po nim **renderowanie obrazu**.



- Zalety:
  - **Stałe** tempo gry (o ile wystarczający sprzęt)
  - Oszczędność **baterii**
- Problemy:
  - Sprzęt nie daje rady osiągnąć tyle FPS => **gra zwalnia**.
  - Wydajny sprzęt => i tak mamy **tyle FPS, co update'ów**.

IMPLEMENTACJA:
Szybkość gry uzależniona od FPS

Podejście na pierwszy rzut oka bardzo dobre,
ale ma **istotne słabe strony**.

- Pozwalamy na renderowanie **tak szybko, jak to jest możliwe**.
- **Każdemu** renderowaniu towarzyszy aktualizacja stanu ze **zmiennym krokiem**.

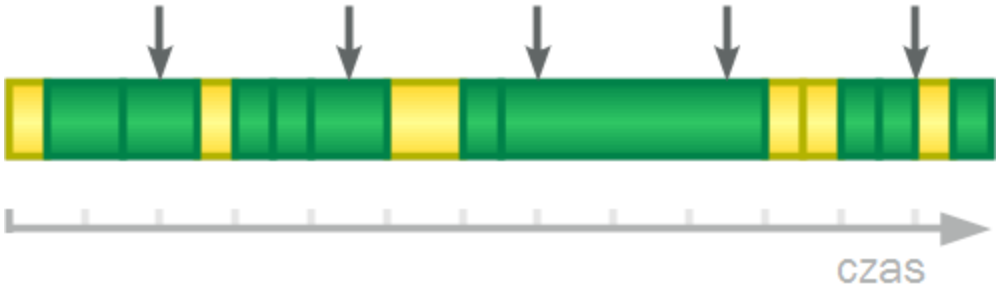
- Zalety:
  - Maksymalna liczba **FPS**
- Problemy:
  - **Nieprzewidywalność** rozgrywki
  - Konieczność aktualizacji **zależnej** od zmiennego czynnika

IMPLEMENTACJA:

Stała szybkość gry i maksimum FPS

Podejście w podstawowej postaci prowadzi do **marnowania** mocy obliczeniowej.

- Rozwinięcie koncepcji stałego kroku aktualizacji
  - Jeśli **mamy jeszcze czas** do zaplanowanej, następnej aktualizacji, to renderujemy **dodatkową klatkę**.
  - Jeśli mamy **opóźnienie**, to wykonujemy dodatkową aktualizację **ze stałym krokiem** żeby nadrobić stratę.



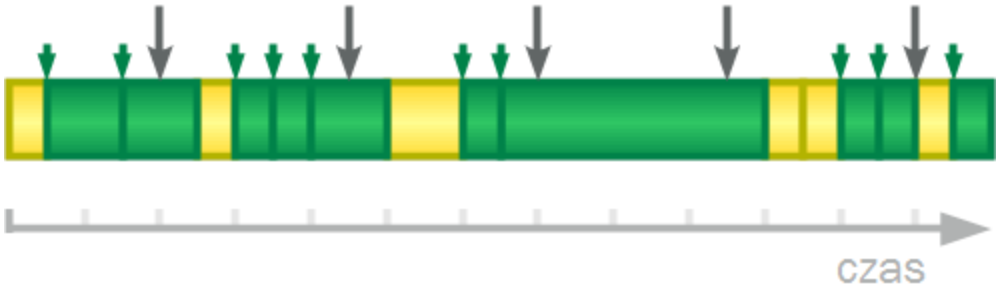
- Zalety:
  - **Rozwiązuje** problem wpływu **długiego czasu renderowania** na częstotliwość aktualizacji.
  - Eliminuje **zbędne oczekiwanie** (czasem to minus).
- Problemy:
  - Jeśli mamy odpowiednio wydajny sprzęt, renderujemy **wiele takich samych klatek**.

IMPLEMENTACJA:

Stała szybkość gry i maksimum FPS + interpolacja

Podejście **zalecane** dla jednowątkowej architektury.

- Aby nie renderować identycznych klatek pomiędzy aktualizacjami, można dokonać **interpolacji**.
  - Wprowadza **opóźnienie** o wielkości jednego cyklu.
- Innym, pokrewnym podejściem jest **predykcja**.
  - Szczególnie dla wartości kontrolowanych przez gracza może wywołać **niepożądane efekty**.

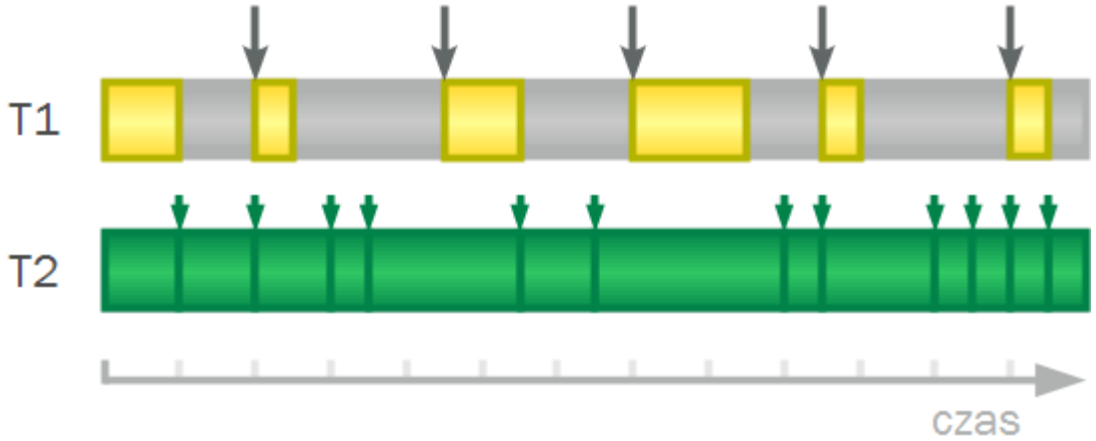


- Zalety:
  - **Stały krok** aktualizacji, **maksimum FPS**.
  - Każda **klatka** jest **inna**.
- Problemy:
  - Nieznaczne **opóźnienie**.

IMPLEMENTACJA:
Zrównoleglony game loop

Konieczne jest korzystanie z **interpolacji** bądź **predykcji**.

- Rozdzielenie aktualizacji i renderowania na **dwa osobne wątki**, które mogą być wykonywane równolegle przez np. procesor wielordzeniowy.



- Zalety:
  - Współbieżne renderowanie i aktualizacja, **niezależne** od siebie.
  - **Stały krok** aktualizacji i **maksymalna liczba FPS**.
- Problemy:
  - Konieczność **uważnego planowania** architektury silnika.

IMPLEMENTACJA:

Wielowątkowość w grach – możliwości i problemy

Nie warto uciekać
od wielowątkowości
w nowoczesnych silnikach.

Współczesny sprzęt daje
ogromne możliwości.
Należy je wykorzystać.

- Jeden **kontekst OpenGL** może w danym momencie być używany tylko przez **jeden wątek**.
  - Komunikacja CPU -> karta graficzna **nie może być** traktowana jako możliwa do zrównoleglenia.
  - Można stworzyć **kilka kontekstów**, które współdzielą dane – **nie jest** to jednak podejście wydajne ani **zalecane**.
- Inne możliwości wykorzystania wielowątkowości:
  - **Odczyt tekstur i zasobów** z dysku odbywający się "w tle".
    - Dysk **nie jest** urządzeniem pozwalającym na odczyt **równoległy** (z wyłączeniem macierzy dyskowych)!
  - Wydajna, zrównoleglona **aktualizacja stanu**:
    - Obliczenia **AI** podzielone na kilka wątków.
    - Symulacja **fizyki**.
    - Systemy **cząsteczkowe**.
  - Odtwarzanie **dźwięku**.

Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

Bartosz Bazyluk

GAME LOOP

Podstawowy element silnika gry komputerowej

Programowanie Gier Komputerowych, Informatyka S1, III Rok

Wydział
Informatyki