Caracas, 06 de Noviembre de 2025

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Introducción a la Computación Gráfica
Semestre 2025-II

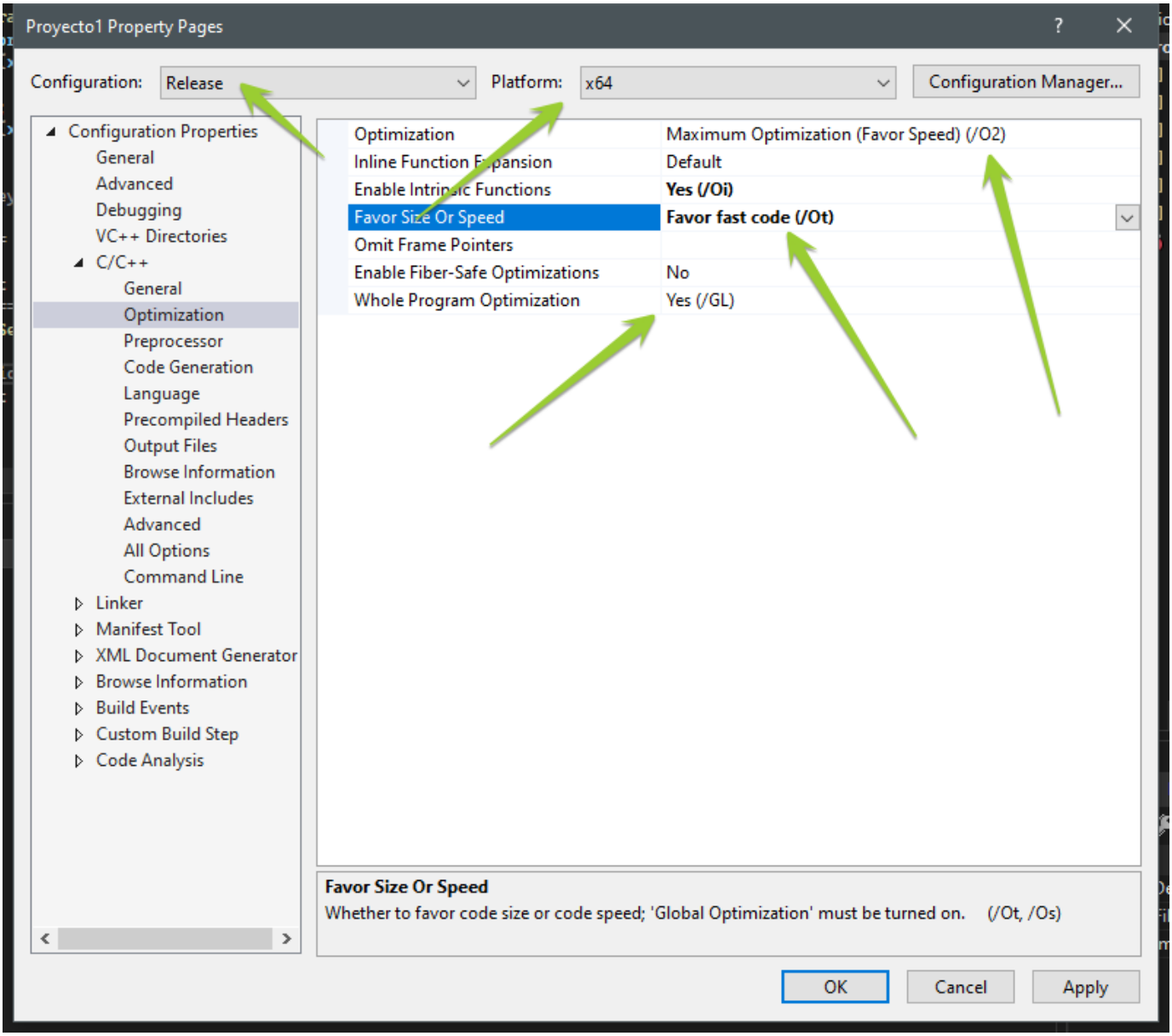
Tarea 2

Ya habrán visto en clase que el algoritmo para despliegue de elipses visto en clase realiza mucho cómputo redundante, aún utilizando aritmética entera. Con un poco de esfuerzo, podemos determinar cómo cambian las expresiones entre iteraciones consecutivas, y podemos notar que basta sumar o restar una constante en cada expresión cada vez que X o Y se incrementan o decrementan en una unidad. Así, podemos escribir un algoritmo (y luego traducirlo como un método en C++) que genere los píxeles de una elipse de manera incremental, lo que significa que dentro de cada ciclo (while, for o do-while) las operaciones serán únicamente sumas, restas, asignaciones y comparaciones, y mejor aún, con valores enteros.

Surge entonces la siguiente pregunta científica: ¿El compilador será capaz de optimizar suficientemente el sub programa original para obtener un código tan o incluso más eficiente que nuestro sub programa "incremental" en tiempo de ejecución?

¿Cómo probar esto?

1. Dado el código base de la tarea #1, implemente el algoritmo de despliegue de elipse original visto en clase, con aritmética entera, sin optimizaciones. Recuerde utilizar "INT64", o simplemente **longlong** para que los enteros sean de 64 bits. Llame a este método drawEllipse1. **(3 puntos)**
2. Re-escriba ese método de manera tal que dentro del ciclo hayan únicamente comparaciones simples (de a lo sumo dos variables), asignaciones, sumas y restas. Llame a este método drawEllipse2. **(5 puntos)**
3. **Programación de Prueba de similitud**: Diseñe e implemente unas pruebas dentro del programa que permita verificar que ambos algoritmos están generando exactamente los mismos píxeles (ni uno más, ni uno menos) en las mismas posiciones, para un conjunto aleatorio de 10.000 elipses. Debe ser una prueba "convincente". Debe emitir un mensaje al usuario indicando si los resultados fueron iguales o no. El usuario debe presionar <enter> para continuar, una vez finalizada la prueba. **(4 puntos)**
4. **Programación de Prueba de eficiencia**: esta prueba continúa luego de presionar <enter> en la prueba anterior. Genere un conjunto de N elipses aleatorias; estos son los atributos que nos interesan: cx, cy, a, b, color. Guarde esos valores vectoriales en una lista. Mida el tiempo total de generar los píxeles de esas N elipses con cada algoritmo, y compare. Haga distintas pruebas variando N, por ejemplo 50.000, 100.000, 150.000… y así sucesivamente hasta 1 millón o más (up to you!), y mida los tiempos para cada N. **(4 puntos)**
5. Para hacer las pruebas de tiempo, no olvide compilar en modo **RELEASE 64 bits**, con optimizaciones de código por el compilador (ver imagen).
6. Haga un informe de a lo sumo 4 páginas, mostrando los códigos para dibujar las elipses, explicando los experimentos realizados, resultados y discusiones (incluyen gráficas). **(4 puntos)**



Enviar la tarea por correo a recs34@gmail.com. Envíe el archivo modificado main.cpp. En caso de requerir modificar PixelRender,*, enviarlos adicionalmente. Adjunte además el PDF o DOCX del informe. Colocar como subject “2025 II – ICG tarea #2 – <nombre y apellido>”. Fecha tope: miércoles 19 de Noviembre 11:59pm.

ÉXITOS – DISFRUTEN !!!

RC/rc