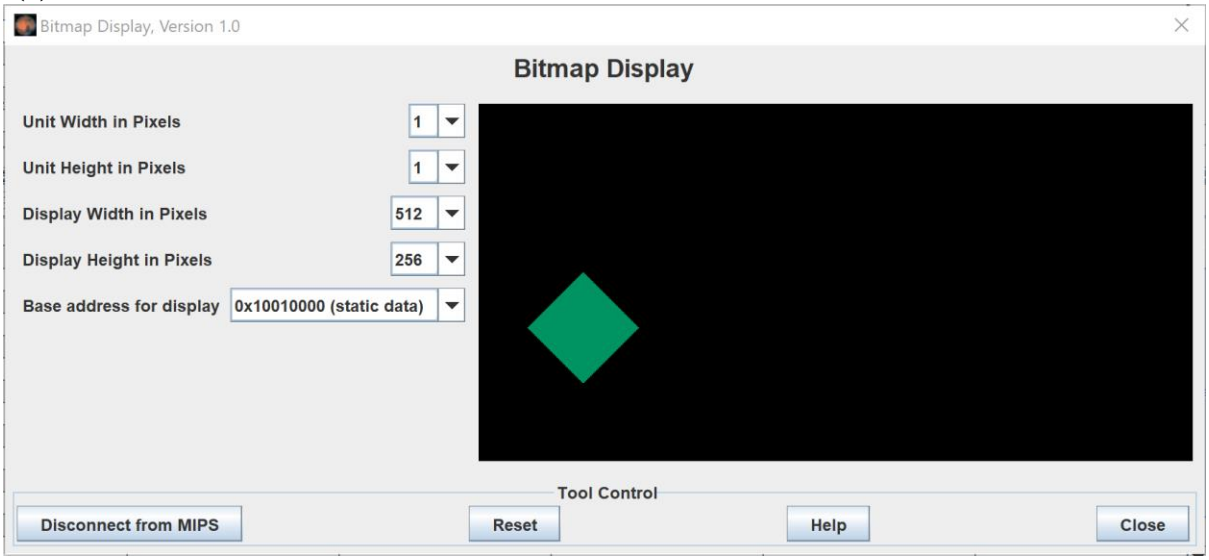
- **Coordenada (x0,y0)**
  - **X0:** Em primeiro momento, decidimos criar uma verificação da valida das posições x0 e y0 fornecidas na entrada do programa. Para x0, verificamos que dado seu valor, a posição inicial estará contida dentro do intervalo de espaço utilizado pelo display do MARS. Prontamente vimos que isto não seria suficiente pois nossas figuras tem uma certa altura e largura a partir do ponto inicial escolhido. Assim, definidos que a largura de nossas figuras será de 80 unidades, portanto modificamos o espaço dentro do qual a posição inicial deve estar contida, de [0, 512] para [0, 432].
  - **Y0:** Analogamente, fizemos a mesma verificação para a posição y0, no entanto, aqui, asseguramos que estivesse presente no intervalo [0,176], onde 176 é o resultado de 256 menos 80 (apesar de que em particular, o triangulo, tem 40 unidades de altura)
  - **ERRO:** Em ambas as coordenadas, no caso de não atenderem aos limites predeterminados, o programa solicita novos valores até que eles estejam adequados (neste caso, uma mensagem de erro será exibida para o usuário)
  - **Metodologia:** A fim de facilitar o entendimento e desenvolvimento, trabalhos com a ideia de que todas as formas requisitas na entrada, estarão dentro de uma quadrado de lado 80. Assim, a coordenada (x0,y0) fornecida pelo usuário, representar a quina superior esquerda deste quadrado. Com o x0 e y0 adequados em mãos, assim como a cor da forma a ser desenhada, seguimos para a logica do nosso programa

- **Logica do programa**
  - Usaremos um espaço definido no .data para termos acesso ao endereço que está o display do MARS. Tal espaço foi chamado de display buffer, e possui o valor 0x80000, que será 512 * 256 * 4. Percebe-se que 512*256 é a área do nosso display e multiplicamos esse valor por 4, pois cada inteiro possui 4 bytes (não teremos problemas em acessar espaços de memória). Calcularemos a distância x0 + largura e y0 + altura, para demarcar até aonde a nossa forma será desenhada. Iniciaremos um loop que irá percorrer linha por linha da nossa forma, inserindo um valor (cor) coluna a coluna em cada endereço de memória. Isso provocará a exibição de cores no display de acordo com a lógica necessária para cada uma das formas. Pararemos o loop das colunas até chegarmos no valor x0 + largura e pararemos o loop das linhas até chegarmos no valor y0 + altura.

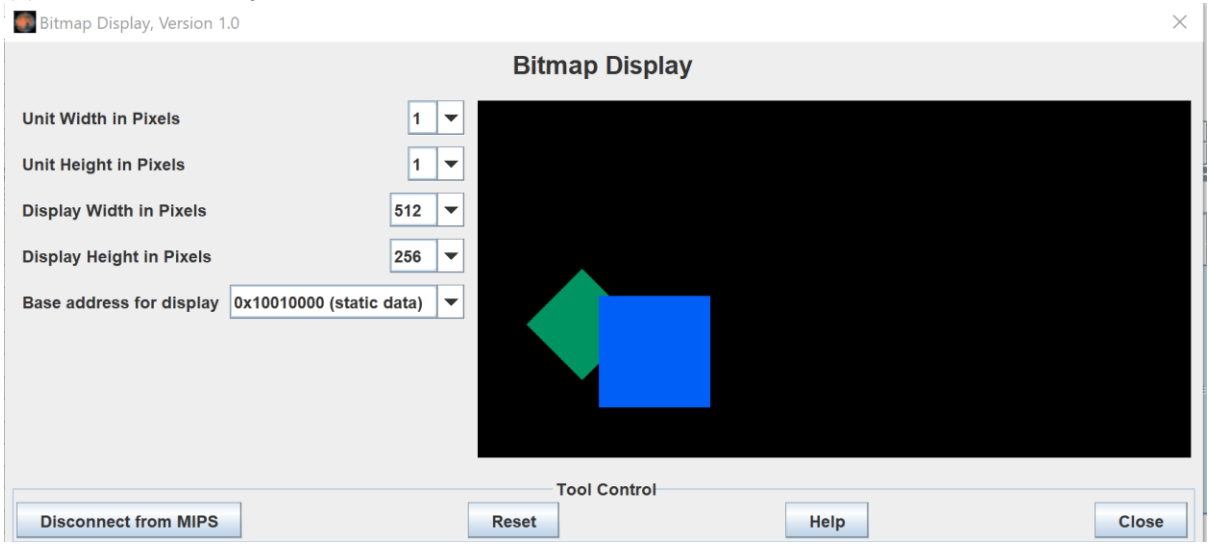
- **Exibição de cada figura:**
  - **Quadrado:** a mais simples das tres formas implementadas. Usando loops, conseguimos desenhar ele sem muitas dificuldades seguindo a lógica simples apresentada acima.
  - **Triangulo:** foi uma modificação do procedimento que desenha o quadrado. Pensamos em tirar uma unidade de cada linha ao longo dos loops. Tendo assim, um formato de triângulo. O triângulo será isósceles, visto que, a base será igual a base do quadrado (80 unidades) e as demais arestas terão um valor de diagonal/2. Para pararmos esse loop, apenas vimos se as arestas laterais se encontram, ou seja, se a1 > a2, por exemplo, teremos que a aresta da esquerda cruzou a aresta da direita.
  - **Losango:** tal como feito entre as formas do quadrado e triangulo, o losango foi novamente uma modificação de procedimentos passados, dessa vez porem, baseada na implementação do triângulo. Para desenharmos um losango, apenas desenhamos um triângulo idêntico ao anterior e partindo do mesmo ponto, desenhamos um triângulo de cabeça para baixo, obtendo assim um losango. Para desenharmos esse triângulo "invertido", apenas ao invés de desenharmos o mesmo para cima (subtraindo 512*4 = 0x800 para irmos para uma linha superior), desenhamos dessa vez para baixo (somando 512*4 = 0x800 para irmos para uma linha inferior).

- **Features não contempladas no programa:**
  - **Círculo:** não conseguimos desenhar um círculo, mas elaboramos algumas ideias para tal. Uma delas, foi usar o valor de pi (3.14159, precisão já suficiente) juntamente com a fórmula analítica de um círculo (sqrt(x^2 - y^2)) para realizar tal feito. Uma outra ideia, foi usar algo similar ao losango, mas de forma mais sutil, resultando em um círculo com bordas pouco arredondas e bastante “pixelizadas”. Como a ideia de fazer um círculo foi posteriormente trocada pela de fazer um losango, deixamos esta implementação de lado.
  - **Leitura hexadecimal:** não conseguimos ler o valor da cor em hexadecimal no MARS presente na definição original do trabalho prático. Pensamos em criar uma função que lê uma string e a partir dela tenta converter o valor lido para um valor em decimal. Como tal tarefa teria uma implementação bastante complicada, mesma foi trocada pela leitura do valor em decimal (tal qual solicitado na nova documentação do trabalho).
- **Testes**
  - A seguir, exemplos do uso do programa no MARS
  - As entradas vêm na sequência:
    - Forma(0 = Sair ; 1 = Quadrado ; 2 = Triangulo ; 3 = Losangulo);
    - Coordenada x_0
    - Coordenada Y_0
    - Cor da forma

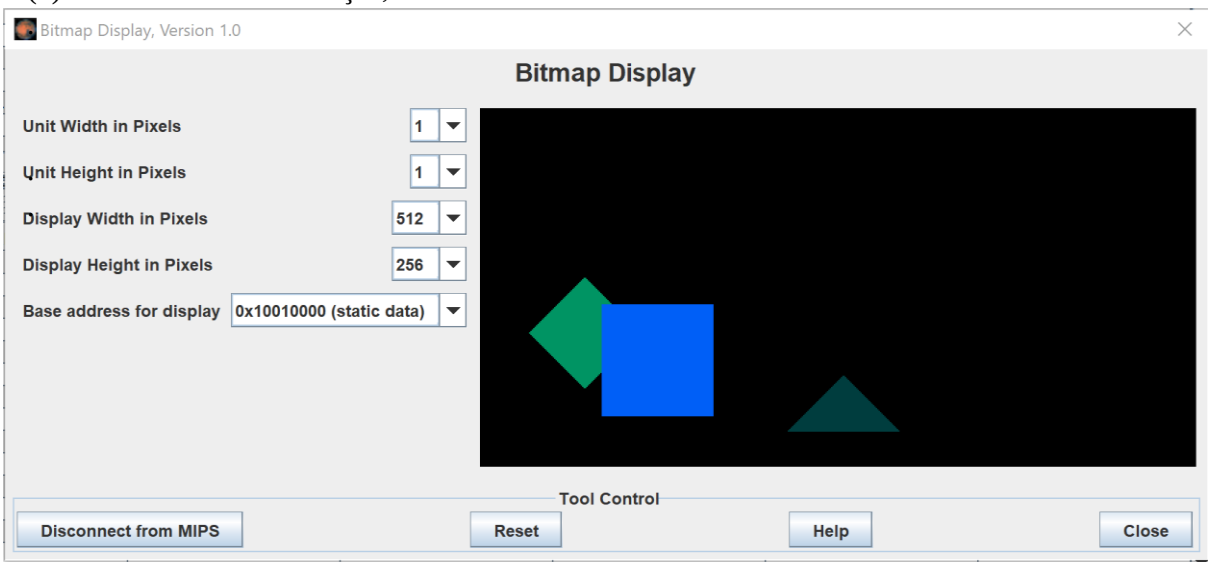
  - (1) Entradas: 3 35 120 37987



  - (2) Na mesma execução do teste 1, entradas: 1 87 140 24567



- (3) Ainda na mesma execução, entradas: 2 220 150 15678



- (4) Numa nova execução, entradas: 3 280 50 129045